T7 Datasheet

High performance multifunction DAQ with USB, Ethernet, and WiFi.

This datasheet covers all T7 variants: T7, T7-OEM, T7-PRO, and T7-PRO-OEM.

Most information in this datasheet applies to all T7 variants. Information about WiFi and the high-resolution ADC (ResolutionIndex = 9-12) only applies to the -Pro variants. There is an OEM section with information specific to the build of OEM versions.

Searching The Datasheet

To search this datasheet you can just use the search box you find on every page, and to further refine your results include "t7" or "t7 datasheet" in your search term. To specifically restrict your search to just this datasheet, include "site:labjack.com/support/datasheets/t7" in your search term.

Navigating the Datasheet using the Table of Contents

An efficient way to navigate this online datasheet is to browse the table of contents to the left. Rather than clicking on all the links to browse, you can click on the small black triangles to expand without reloading the whole page.

Offline Datasheet

If you are looking at a PDF, hardcopy, or other downloaded offline version of this datasheet, realize that it is possibly out-of-date as the original is an online document. Also, this datasheet is designed as online documentation, so the formatting of an offline version is often less than ideal.

To create an offline version of this entire document, click "Print all" towards the upper-right of this page (allow 30+ seconds). To make an offline version of a particular section go to that page and click "Print all" towards the upper-right of that page. Doing so converts this page and all sub-pages to a single HTML page. If you need to translate the document, do that now. To produce a PDF, use the print option in your browser and use whatever print-to-PDF option you might have available (e.g. PrimoPDF).

Rather than downloading, though, we encourage you to use this web-based documentation. Some advantages:

- We can quickly improve and update content.
- Click-able links to further or related details throughout the online document.
- The site search includes the datasheet, forum, and all other resources at labjack.com. When you are looking for something try using the site search.
- For support, try going to the applicable datasheet page and post a comment. When appropriate we can then immediately add/change content on that page to address the question.

Occasionally we export a PDF and attach it to this page (below).

Datasheet
File attachment:
T7-Datasheet-Export-20150217.pdf

Preface: Warranty, Liability, Compliance

For the latest version of this and other documents, go to www.labjack.com.

Copyright 2013, LabJack Corporation

Warranty:

The LabJack T7 is covered by a 1 year limited warranty from LabJack Corporation, covering this product and parts against defects in material or workmanship. The LabJack can be damaged by misconnection (such as connecting 120 VAC to any of the screw terminals), and this warranty does not cover damage obviously caused by the customer. If you have a problem, contact support@labjack.com for return authorization. In the case of warranty repairs, the customer is responsible for shipping to LabJack Corporation, and LabJack Corporation will pay for the return shipping.

Limitation of Liability:

LabJack designs and manufactures measurement and automation peripherals that enable the connection of a PC to the real-world. Although LabJacks have various redundant protection mechanisms, it is possible, in the case of improper and/or unreasonable use, to damage the LabJack and even the PC to which it is connected. LabJack Corporation will not be liable for any such damage.

Except as specified herein, LabJack Corporation makes no warranties, express or implied, including but not limited to any implied warranty or merchantability or fitness for a particular purpose. LabJack Corporation shall not be liable for any special, indirect, incidental or consequential damages or losses, including loss of data, arising from any cause or theory.

LabJacks and associated products are not designed to be a critical component in life support or systems where malfunction can reasonably be expected to result in personal injury. Customers using these products in such applications do so at their own risk and agree to fully indemnify LabJack Corporation for any damages resulting from such applications.

LabJack assumes no liability for applications assistance or customer product design. Customers are responsible for their applications using LabJack products. To minimize the risks associated with customer applications, customers should provide adequate design and operating safeguards.

Reproduction of products or written or electronic information from LabJack Corporation is prohibited without permission. Reproduction of any of these with alteration is an unfair and deceptive business practice.

Conformity Information (FCC, CE, RoHS):

See the Conformity Page and the text below:

FCC PART 15 STATEMENTS:

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense. The end user of this product should be aware that any changes or modifications made to this equipment without the approval of

the manufacturer could result in the product not meeting the Class A limits, in which case the FCC could void the user's authority to operate the equipment.

Declaration of Conformity:

Manufacturers Name: LabJack Corporation
Manufacturers Address: 3232 S Vance St STE 100, Lakewood, CO 80227, USA

Declares that the product

Product Name: LabJack T7 (-Pro)
Model Number: LJT7 (-Pro)

conforms to the following Product Specifications:

EMC Directive: 2004/104/EEC

EN 55011 Class A
EN 61326-1: General Requirements

and is marked with CE

RoHS2:

The T7 (-Pro) is RoHS compliant to Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

REACH:

The T7 (-Pro) is REACH compliant. REACH Product Compliance Program has been implemented in accordance with Regulation No. 1907/2006 of the European Parliament and the Council of 18 December 2006. LabJack Corporation does not currently have a direct REACH obligation to pre-register substances. LabJack's REACH Product Compliance is determined by a certification from our supply chain. LabJack products are deemed to be REACH compliant when they do not contain Substances of Very High Concern (SVHCs) beyond the specified concentration limits of less than 0.1% by weight as outlined in REACH 1907/2006/EU regulation.

CFM:

LabJack Corporation does not knowingly use these minerals or any by-products, as specified by the Conflict Minerals Trade Act.

Compliance Information for the WiFi module in the T7-Pro and T7-Pro-OEM

FCC:

Contains FCC ID: T9J-RN171
This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy, and if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:
• Reorient or relocate the receiving antenna.
• Increase the separation between the equipment and receiver.
• Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
• Consult the dealer or an experienced radio/TV technician for help.

To satisfy FCC RF Exposure requirements for mobile and base station transmission devices, a separation distance of 20 cm or more should be maintained between the antenna of this device and persons during operation. To ensure compliance, operation at closer than this distance is not recommended. The antenna(s) used for this transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

Canada:

Contains transmitter module IC: 6514A-RN171

This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes: (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

Under Industry Canada regulations, this radio transmitter may only operate using an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (e.i.r.p.) is not more than that necessary for successful communication.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire à l'établissement d'une communication satisfaisante.

This radio transmitter (identify the device by certification number, or model number if Category II) has been approved by Industry Canada to operate with the antenna types listed below with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in

this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire à l'établissement d'une communication satisfaisante.

Preface

1.0 Device Overview

This document contains device-specific information for the following devices:

- T7
- T7-Pro
- T7-OEM
- T7-Pro-OEM

This family introduces a new line of high-quality analog and Ethernet data acquisition hardware, with the main traditional advantage of all LabJack data acquisition hardware, namely, high performance and rich feature set at a competitive price point. These features make the T series a logical choice for many high-performance applications, where Ethernet, WiFi, and cost are primary considerations.

1.1 Core Features

Analog I/O

- 14 Analog Inputs (16-18+ Bits Depending on Speed), expand to 84 with Mux80
- Single-Ended Inputs (14) or Differential Inputs (7)
- Instrumentation Amplifier Inputs
- Software Programmable Gains of x1, x10, x100, and x1000
- Analog Input Ranges of ±10, ±1, ±0.1, and ±0.01 Volts
- 2 Analog Outputs (12-Bit, ~0-5 Volts)

Digital I/O

- 23 Digital I/O
- Supports SPI, I2C, 1-Wire and Asynchronous Serial Protocols (Master Only)
- Supports Software or Hardware Timed Acquisition
- Maximum Input Stream Rate of 100 kHz (Depending on Resolution)
- Capable of Command/Response Times Less Than 1 Millisecond

Digital I/O Extended Features

- Simple PWM Output (1-32 bit)
- PWM Output w/ phase control
- Pulse Output w/ phase control
- Positive edge capture
- Negative edge capture
- PWM measure
- Edge capture & compare
- High speed counter
- Software counter
- Software counter w/ debounce
- Quadrature Input
- Easy Frequency Input

Analog Input Extended Features

- User Defined Slope & Offset
- Thermocouple E, J, K, R, and T calculations
- RTDs

Other highlights

- Built-In CJC Temperature Sensor
- Watchdog system
- Field Upgradable Firmware
- Programmable Startup Defaults
- LJTick Compatible

Fixed Current Outputs

- 200 µA
- 10 µA

1.2 Family Variants Info

T7 vs T7-Pro

The T7-Pro has all features of the normal T7, with the following added:

- Wireless Ethernet 802.11b/g
- 24-bit Low-Speed ADC for 22-Bit Effective Resolution
- Battery-Backed Real Time Clock to assist Scripts.

Also see the block diagram in the hardware overview section.

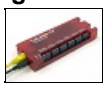
T7-OEM and T7-Pro-OEM

There is also an OEM version of the T7 and T7-Pro. The OEM versions are the same in terms of features, but the enclosure, and most connectors are not installed on the OEM versions, which allows customers to easily configure as needed. See Appendix A - OEM Versions for details.

2.0 Installation

1. First install LabJack software and driver bundle based on your operating system.

   T7/Digit Devices

	Windows Installer	79.09 MB	2015-03-10 21:25	details
	Mac OS X Package	2.86 MB	2015-02-10 12:56	details
	Linux 32-bit Package	1.44 MB	2014-07-29 15:57	
	Linux 64-bit Package	1.45 MB	2014-07-29 15:56	
	Linux ARMv7 Package	1.35 MB	2014-10-02 17:27	details

2. Connect the T7 to the local computer via USB.
3. Proceed through any steps to add new hardware.
4. If using Windows, open Kipling (installed with package above). Utility apps for other operating systems are still under development.
5. Use the dashboard in Kipling to view analog inputs, digital I/O, DAC outputs, etc.
6. Go to quickstart page to see more about Kipling and its use with the T7.

3.0 Communication

Modbus TCP is the protocol used by all connections on the T7(USB, Ethernet, WiFi). All important values and data from the device

can be read and/or written by using the associated Modbus register(s). Thus, the process for reading the serial number, an analog input, or a waveform is all functionally the same, you simply provide a different address. There are two main ways to communicate with a T7 using Modbus TCP.

Communication Options

High-level LJM library

Among other useful features, this cross-platform library allows users to access registers by name, such as "AIN4" for analog input 4. Most people will use the LJM library since they're familiar with writing code, and want to integrate a T7 into an existing software framework.

Conceptual workflow:

1. Find example code/wrappers for your desired programming language.
2. Use the LJM_Open() function to open a connection to the T7.
3. Perform reads and writes to Modbus registers using LJM_eReadName() or LJM_eWriteName().
4. Use the Close() function to close the connection.

Direct Modbus TCP, Clients

It is easy to integrate a T7 over Ethernet or WiFi into standard COTS Modbus software platforms, since the T7 is directly compatible. People who already use Modbus software will find this option convenient. Some COTS Modbus software is very powerful, and will save users the time and money required to develop their own software.

Conceptual workflow:

1. Configure the power-up-default registers on the T7 using the Kipling software program. Change Ethernet/WiFi IP settings, any relevant analog input settings, etc. '..._DEFAULT' registers indicate that they are power-up-defaults.
2. Open COTS Modbus program.
3. Specify the Modbus registers by address, such as 8, for AIN4. Find applicable registers with the register look-up tool, or by referencing the datasheet etc.
4. See data directly from the T7 in COTS software.

Communication Speed Considerations

There are two alternate methods for data transfer to occur, command response is the lowest latency, and streaming offers the highest data throughput, the following sections provide more detail.

Command-Response

This is the default behavior for communication with a device, and most people find the data throughput satisfactory. Direct Modbus interactions will always use command-response. The high-level LJM library also uses command-response.

Communication is initiated by a command from the host which is followed by a response from the device. In other words, data transfer is software-paced. Command-response is generally used at 1000 scans/second or slower and is generally simpler than stream mode.

Command-response mode is generally best for minimum-latency applications such as feedback control. By latency here we mean the time from when a reading is acquired to when it is available in the host software. A reading or group of readings can be acquired in times on the order of a millisecond. See Appendix A-1 for details on c-r data rates.

Stream Mode

Stream mode is generally best for maximum-throughput applications. However, streaming is not recommended for feedback control operations, due to the latency in data recovery. Data is acquired very fast, but to sustain the fast rates it must be buffered and moved from the device to the host in large chunks. Streaming is only available through the high-level LJM library, and is not possible using direct Modbus communication.

Stream mode is a continuous hardware-paced input mode where a list of addresses is scanned at a specified scan rate. The scan rate specifies the interval between the beginning of each scan. The samples within each scan are acquired as fast as possible. As samples are collected automatically by the device, they are placed in a buffer on the device, until retrieved by the host. Stream mode is generally used when command-response is not fast enough. Stream mode is not supported on the hi-res converter (resolutions 9-12 not supported in stream).

For example, a typical stream application might set up the device to acquire a single analog input at 100,000 samples/second. The device moves this data to the host in chunks of 25 samples each. The LJM library moves data from the USB host memory to the software memory in chunks of 2000 samples. The user application might read data from memory once a second in a chunk of 100,000 samples. The computer has no problem retrieving, processing, and storing, 100k samples once per second, but it could not do that with a single sample 100k times per second. See Appendix-A-1 for details on stream mode data rates.

Command-response can be done while streaming, but streaming needs exclusive control of the analog input system so analog inputs (including the internal temperature sensor) cannot be read via command-response while a stream is running.

3.1 Modbus Map

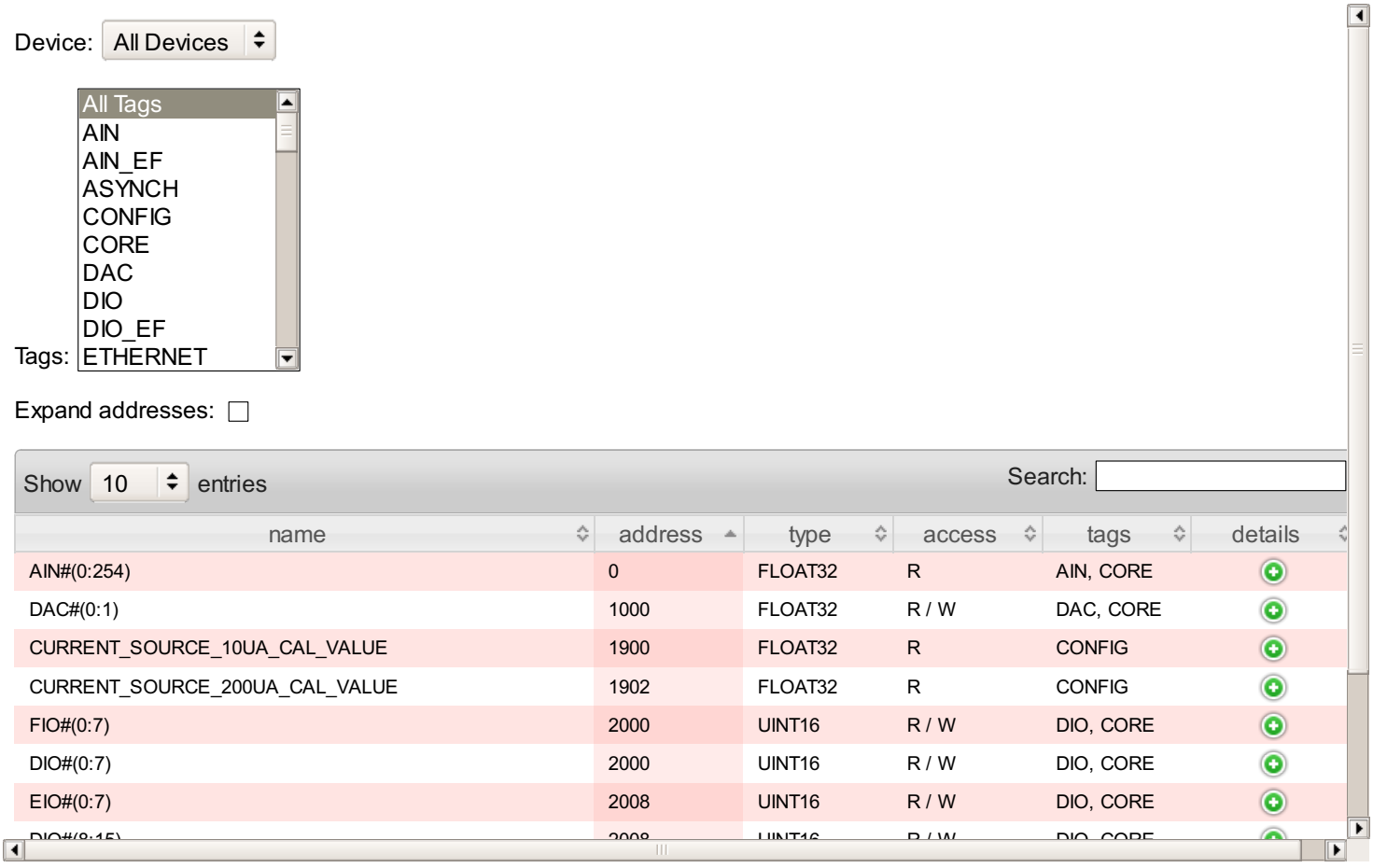
This utility is used to find Modbus registers that pertain to the T7, simply select the T7 from the device filter, and then narrow down results using either the search tool, or the tag filters. All of these registers use command-response, unless specified as STREAM.

http://labjack.com/support/modbus/map

We distribute a constants file called "ljm_constants.json" that defines all information about the Modbus register map. The dynamic filter & search tool below pulls data from that JSON file.

- Name: A string name that can be used with the LJM library to access each register.
- Address: The starting address of each register. Valid through LJM or with direct Modbus.
- Details: Click to get text pulled from the description field in the JSON.
- Type: Specifies the datatype, which also tells you how many registers each value uses.
- Access: Read-only, write-only, or read & write.
- Tags: Used to associate registers with particular functionality. Useful for filtering.

For a U3/U6 with firmware less than 2.0, or for the UE9, see the deprecated Modbus system called UD Modbus.



3.2 Stream Mode

The highest input data rates are obtained in stream mode. Stream is a continuous hardware timed input mode where a list of channels/addresses is scanned at a specified scan rate. The scan rate specifies the interval between the beginning of each scan. The samples within each scan are acquired as fast as possible.

The **max sample rate** of the T7 is 100 ksamples/second. This is achievable for any single-address stream, but for a multi-address stream is only true if ResolutionIndex = 0 or 1 and Range = +/-10V for all analog inputs. The **max scan rate** depends on how many addresses you are sampling per scan:

Sample => A reading from one channel/address.
Scan => One reading from all channels/addresses in the scan list.
SampleRate = NumAddresses * ScanRate

For example, if streaming 5 channels at ResolutionIndex=0 and all at Range=+/-10V, the max scan rate is 20 kscans/second (calculated from 100 ksamples/second divided by 5).

Ethernet is capable of the fastest stream rates. USB is typically a little slower than Ethernet, and WiFi is much slower. For more information on speeds see the Data Rates Appendix.

Stream mode is not supported on the hi-res converter (resolutions 9-12 not supported in stream).

As samples are collected, they are placed in a FIFO buffer on the T7, until retrieved by the host. The size of the buffer is variable and can be set to a maximum of 32764 bytes. Each data packet has various measures to ensure the integrity and completeness of the data received by the host.

Data can be sent to the host in two different ways. The first is “Spontaneous.” In spontaneous mode packets are automatically sent to the host as soon as there is enough data to fill a packet. The packet size is adjustable, see the register definitions below. The second is “Command-Response” (CR). In CR mode the stream data is stored in the T7’s buffer and must be read out using a command. CR mode is useful for when the connection is unreliable.

The T7 uses a feature called auto-recovery. If the buffer overflows, the T7 will continue streaming but discard data until the buffer is emptied, and then data will be stored in the buffer again. The T7 keeps track of how many scans are discarded and reports that value. Based on the number of scans discarded, the LJM driver adds the proper number of dummy samples (-9999.0) such that the correct timing is maintained. Auto-recover will only work if the first channel in the scan is an analog channel.

The scan rate is generated from the internal crystal oscillator. Alternatively, the scan rate can be a division of an external clock provided on CIO3.

The timing between items in the scan list is controlled by several factors. Timing pulses are generated on SPC so that the timing can be measured. Pulses on SPC are as follows. Falling edge at the start of a scan. Rising edge at the start of a sample. Falling edge at the end of a sample. Rising edge at the end of a scan.

Registers that can be streamed are marked **"streamable":true** in ljm_constants.json and include:

- AIN#
- FIO_STATE
- EIO_STATE
- CIO_STATE
- MIO_STATE
- FIO_EIO_STATE
- EIO_CIO_STATE
- DIO#(0:22)_EF_READ_A
- DIO#(0:22)_EF_READ_A_AND_RESET
- DIO#(0:22)_EF_READ_B
- CORE_TIMER
- SYSTEM_TIMER_20HZ
- STREAM_DATA_CAPTURE_16

Stream data is transferred as 16-bit values only. In the normal case of an analog input such as AIN0, the 16-bit binary value is actually is what is transferred and LJM converts it to a float on the host using the cal constants that LJM reads in before starting the stream. Some registers in the list above (e.g. DIO4_EF_READ_A) have 32-bits data. When streaming a register that produces 32-bit data, the lower 16-bits (LSW) will be returned and the upper 16-bits (MSW) will be saved in STREAM_DATA_CAPTURE_16. To get the full 32-bit value, add STREAM_DATA_CAPTURE_16 to the stream scan list after any applicable register, then combine the two values in software (LSW + 65536*MSW). Note that it would not be unusual to put STREAM_DATA_CAPTURE_16 in multiple locations in the scan list.

The following describes the Modbus registers related to stream mode. Note that the LJM library provides special functions for stream, so most users will not use the following registers.

Stream Configuration

Name	Start Address	Type	Access	Default
STREAM_SCANRATE_HZ	4002	FLOAT32	R/W	0
STREAM_NUM_ADDRESSES	4004	UINT32	R/W	0
STREAM_SAMPLES_PER_PACKET	4006	UINT32	R/W	0
STREAM_SETTLING_US	4008	FLOAT32	R/W	0
STREAM_RESOLUTION_INDEX	4010	UINT32	R/W	0
STREAM_BUFFER_SIZE_BYTES	4012	UINT32	R/W	0
STREAM_AUTO_TARGET	4016	UINT32	R/W	0

STREAM_NUM_SCANS	4020	UINT32	R/W	0
STREAM_ENABLE	4990	UINT32	W	
STREAM_SCANLIST_ADDRESS#(0:127)	4100	UINT32	R/W	0

STREAM_SCANRATE_HZ
The number of times per second that all channels in the scanlist will be read. SampleRate=NumChannels*ScanRate. Has no effect when using and external clock.

STREAM_NUM_ADDRESSES
The number of entries in the scanlist

STREAM_SAMPLES_PER_PACKET
Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

STREAM_SETTLING_US
Time in microseconds to allow signals to settle after switching the mux. Default=0 equates to 10 us if sample rate <80k, and 6 us for higher sample rates. Max is 5000.

STREAM_RESOLUTION_INDEX
Index specifying the resolution of the data. High settings will have lower max speeds.

STREAM_BUFFER_SIZE_BYTES
Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2, as it takes 2 bytes to hold 1 sample. Changes while stream is running do not affect the currently running stream.

STREAM_AUTO_TARGET
Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

STREAM_NUM_SCANS
The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Max number of samples is STREAM_BUFFER_SIZE_BYTES/2. Max number of scans is (STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES.

STREAM_ENABLE
Write 1 to start stream. Write 0 to stop stream.

STREAM_SCANLIST_ADDRESS#(0:127)
A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Names	Addresses
STREAM_SCANLIST_ADDRESS0, STREAM_SCANLIST_ADDRESS1, STREAM_SCANLIST_ADDRESS2, Show All	4100, 4102, 4104, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

3.2.1 Stream-Out (Advanced)

Stream-Out is a set of streamable registers that move data from a buffer to an output. The output can be digital I/O (DIO) or a DAC. The buffer can be read linearly to generate a irregular waveform or in a looping mode to generate a periodic waveform. The T7 can output **up to 4 waveforms** using stream out.

Targets

The following target list represents the I/O on the device that can be configured to output a waveform using stream out. Basically the list includes the analog and digital output lines.

```
DAC0
DAC1
FIO_STATE
FIO_DIRECTION
EIO_STATE
EIO_DIRECTION
CIO_STATE
CIO_DIRECTION
MIO_STATE
MIO_DIRECTION
```

Stream-Out

Name	Start Address	Type	Access	Default

STREAM_OUT#(0:3)	4800	UINT16	R	0

STREAM_OUT#(0:3)
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0:127) to trigger stream out updates. This register returns null data when read via command-response.

Names	Addresses
STREAM_OUT0, STREAM_OUT1, STREAM_OUT2, Show All	4800, 4801, 4802, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

-->

Configuration

Configuration will set the buffer size and target. The target specifies which physical I/O to use. Data in the buffer will be output onto the target I/O as a generated waveform. Configuration can be done before or after stream has started.

Stream-Out Configuration

Name	Start Address	Type	Access	Default
STREAM_OUT#(0:3)_TARGET	4040	UINT32	R/W	0
STREAM_OUT#(0:3)_BUFFER_SIZE	4050	UINT32	R/W	0
STREAM_OUT#(0:3)_ENABLE	4090	UINT32	R/W	0

STREAM_OUT#(0:3)_TARGET
Channel that data will be written to.

Names	Addresses
STREAM_OUT0_TARGET, STREAM_OUT1_TARGET, STREAM_OUT2_TARGET, Show All	4040, 4042, 4044, Show All

STREAM_OUT#(0:3)_BUFFER_SIZE
Size of the buffer in bytes. Should be at least twice the size of updates that will be written. Max is 16384.

Names	Addresses
STREAM_OUT0_BUFFER_SIZE, STREAM_OUT1_BUFFER_SIZE, STREAM_OUT2_BUFFER_SIZE, Show All	4050, 4052, 4054, Show All

STREAM_OUT#(0:3)_ENABLE
Write 1 to enable, 0 to disable.

Names	Addresses
STREAM_OUT0_ENABLE, STREAM_OUT1_ENABLE, STREAM_OUT2_ENABLE, Show All	4090, 4092, 4094, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

STREAM_OUT#(0:3)_BUFFER_SIZE – Number of bytes that will be allocated to this Stream-Out's buffer. Must be a power of 2.
STREAM_OUT#(0:3)_ENABLE – 1 = Stream-Out# on. 0 = Stream-Out# off.
-->

Buffers

Each Stream-Out has its own buffer. Data is loaded into the buffer by writing to the appropriate buffer register. Output waveform data points are stored in the buffer as 16-bit values, so values greater than 16-bits will be converted automatically before being stored in the buffer. Use only one buffer per STREAM_OUT channel. For outputting an analog waveform(DAC output), write an array of floating point numbers to the STREAM_OUT#(0:3)_BUFFER_F32 register. For outputting a digital waveform, pass an array of integer 0 or 1 values to the STREAM_OUT#(0:3)_BUFFER_U16 register.

Stream-Out Buffers

Name	Start Address	Type	Access	Default
STREAM_OUT#(0:3)_BUFFER_U16	4420	UINT16	W	0
STREAM_OUT#(0:3)_BUFFER_U32	4410	UINT32	W	0
STREAM_OUT#(0:3)_BUFFER_F32	4400	FLOAT32	W	0
STREAM_OUT#(0:3)_BUFFER_STATUS	4080	UINT32	R	0

STREAM_OUT#(0:3)_BUFFER_U16

Data destination when sending 16-bit integer data.

Names	Addresses
STREAM_OUT0_BUFFER_U16, STREAM_OUT1_BUFFER_U16, STREAM_OUT2_BUFFER_U16, Show All	4420, 4421, 4422, Show All

STREAM_OUT#(0:3)_BUFFER_U32

Data destination when sending 32-bit integer data. Takes up 2 bytes of the stream out buffer.

Names	Addresses
STREAM_OUT0_BUFFER_U32, STREAM_OUT1_BUFFER_U32, STREAM_OUT2_BUFFER_U32, Show All	4410, 4412, 4414, Show All

STREAM_OUT#(0:3)_BUFFER_F32

Data destination when sending floating point data. Takes up 2 bytes of the stream out buffer.

Names	Addresses
STREAM_OUT0_BUFFER_F32, STREAM_OUT1_BUFFER_F32, STREAM_OUT2_BUFFER_F32, Show All	4400, 4402, 4404, Show All

STREAM_OUT#(0:3)_BUFFER_STATUS

The number of entries in the buffer that are not currently being used.

Names	Addresses
STREAM_OUT0_BUFFER_STATUS, STREAM_OUT1_BUFFER_STATUS, STREAM_OUT2_BUFFER_STATUS, Show All	4080, 4082, 4084, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

STREAM_OUT#(0:3)_BUFFER_U32
STREAM_OUT#(0:3)_BUFFER_F32
STREAM_OUT#(0:3)_BUFFER_STATUS – The free space in the buffer in data points.
-->

Once the waveform data points are stored, the next step is to configure the STREAM_OUT#(0:3)_LOOP_SIZE and STREAM_OUT#(0:3)_SET_LOOP parameters to establish if/when the waveform repeats or stops, or if the waveform should be clipped relative

Stream-Out Waveform Periodicity

Name	Start Address	Type	Access	Default
STREAM_OUT#(0:3)_LOOP_SIZE	4060	UINT32	R/W	0
STREAM_OUT#(0:3)_SET_LOOP	4070	UINT32	W	0

STREAM_OUT#(0:3)_LOOP_SIZE

The number of value that will be repeated after reaching the end of supplied data.

Names	Addresses
STREAM_OUT0_LOOP_SIZE, STREAM_OUT1_LOOP_SIZE, STREAM_OUT2_LOOP_SIZE, Show All	4060, 4062, 4064, Show All

STREAM_OUT#(0:3)_SET_LOOP

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different Stream-Out channel is set to Synch. 3=Synch. This Stream-Out# as well as any Stream-Outs set to synch will start using new data immediately.

Names	Addresses
STREAM_OUT0_SET_LOOP, STREAM_OUT1_SET_LOOP, STREAM_OUT2_SET_LOOP, Show All	4070, 4072, 4074, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

STREAM_OUT#(0:3)_SET_LOOP – Controls when new data sets are used.
1 – Use new data immediately.
2 – Wait for synch. New data will not be used until a different Stream-Out channel is set to Synch.
3 – Synch. This Stream-Out# as well as any Stream-Outs set to synch will start using new data immediately.
-->

Example

This example demonstrates how to configure DAC0 to output an analog waveform that resembles a triangle wave, and also quickly measure two analog inputs AIN0 and AIN2 in streaming context.

Configuration steps specific to stream-out

1. STREAM_OUT0_ENABLE = 0 –> Turn off just in case it was already on.
2. STREAM_OUT0_TARGET = 1000 –> Set the target to DAC0.
3. STREAM_OUT0_BUFFER_SIZE = 512 –> A buffer to hold up to 256 values.
4. STREAM_OUT0_ENABLE = 1 –> Turn on Stream-Out0.

General stream configuration

1. STREAM_SCANLIST_ADDRESS0= AIN0 –> Add AIN0 to the list of things to stream in.
2. STREAM_SCANLIST_ADDRESS1= STREAM_OUT0 –> Add STREAM_OUT0 (DAC0 is target) to the list of things to stream out.
3. STREAM_SCANLIST_ADDRESS2= AIN2 –> Add AIN2 to the list of things to stream in.
4. STREAM_ENABLE = 1 –> Start streaming.

Note that there are some other settings related to streaming the analog inputs, but those are covered under the section for stream mode.

Load the waveform data points

The following data points have been chosen to produce the triangle waveform: 0.5V, 1V, 1.5V, 1V, so the next step is to write these datum to the appropriate buffer. Because it is a DAC output(floating point number), use the STREAM_OUT0_BUFFER_F32 register.

1. STREAM_OUT0_BUFFER_F32 –> Write the four values one at a time or as an array.
2. STREAM_OUT0_LOOP_SIZE = 4 –> Loop four values.
3. STREAM_OUT0_SET_LOOP = 1 –> Begin using new data set immediately.

Observe result with stream mode

Every time the stream is run, AIN0 is read, then DAC0 is updated with a data point from Stream-Out0's buffer, then AIN2 is read. Thus, the streaming speed dictates the frequency of the output waveform.

Sequential Data

Once a sequence of values has been set via the STREAM_OUT#_SET_LOOP register, that sequence of values will loop and only be interrupted at the end of the sequence. Therefore, to have stream out continuously output a sequence of values that is larger than the size of one stream out buffer, probably the easiest way to do so is to:

1. Start by dividing the stream out buffer into 2 halves,

2. Write one half of the buffer with your sequential data,

3. In a loop, every time the STREAM_OUT#_BUFFER_STATUS reads as being half full/empty, write another half buffer-worth of values.

Note that the buffer is a circular array, so you could end up overwriting values if you're not careful.

Here's an example:

Stream out buffer is 512 bytes, divide that by 2 to get the number of samples the buffer can hold => 256 samples

256 samples divided by 2 to get the "loop" size, AKA the set-of-data-to-be-written-at-a-time size => 128 samples

Write 128 samples:

Write 128 to STREAM_OUT0_LOOP_SIZE

Write 128 samples to STREAM_OUT0_BUFFER_F32 (This should probably be done by array write, which is much faster than writing values individually.)

Write 1 to STREAM_OUT0_SET_LOOP

Loop while you have more sequential data to write:

Read STREAM_OUT0_BUFFER_STATUS

If STREAM_OUT0_BUFFER_STATUS is 128 or greater, write the next 128 samples

Sleep for something like (1 / scanRate) to prevent unnecessary work for the hardware

3.2.1 Stream-Out

3.2.2 Low-Level Streaming

Introduction

Streaming is a data input/output mode of the T7 that collects data at a constant rate and is capable of fast data-collection and data-output rates (approximately 100 kHz). Stream mode is complicated but can easily be executed using the LJM stream functions. The rest of this section is about low-level stream protocol. For details on how to configure stream, see Stream Mode.

The Basics

There are three input/output combinations of stream mode:

Stream-in: The T7 collects data and streams it to the host.

Stream-out: The T7 does not collect data but streams it out. (See Stream Out)

Stream-in-out: The T7 collects data and streams it to the host. It also streams data out.

The stream channels determine which of these modes are used. Streamable channels may be either stream-in or stream-out.

Executing stream mode involves the following:

- Stream setup
- Stream start
- Stream-in data collection
- Stream-out buffer updates (See Stream Out)
- Stream stop

Note that spontaneous stream mode is the mode where the T7 sends data without a Modbus command request. Spontaneous stream mode is not compatible with WiFi T7 connections. WiFi T7 stream requires Command-Response stream. Please leave a comment if you need low-level documentation for WiFi Command-Response stream.

Setup

Stream setup requires configuration of the registers that LJM_eStreamStart takes care of and some others:

Stream Configuration

Name	Start Address	Type	Access	Default
STREAM_SCANRATE_HZ	4002	FLOAT32	R/W	0
STREAM_NUM_ADDRESSES	4004	UINT32	R/W	0
STREAM_SAMPLES_PER_PACKET	4006	UINT32	R/W	0
STREAM_SETTLING_US	4008	FLOAT32	R/W	0
STREAM_RESOLUTION_INDEX	4010	UINT32	R/W	0
STREAM_BUFFER_SIZE_BYTES	4012	UINT32	R/W	0
STREAM_CLOCK_SOURCE	4014	UINT32	R/W	0
STREAM_AUTO_TARGET	4016	UINT32	R/W	0
STREAM_NUM_SCANS	4020	UINT32	R/W	0
STREAM_EXTERNAL_CLOCK_DIVISOR	4022	UINT32	R/W	0
STREAM_ENABLE	4990	UINT32	W	

STREAM_SCANLIST_ADDRESS#(0:127)	4100	UINT32	R/W	0

STREAM_SCANRATE_HZ
The number of times per second that all channels in the scanlist will be read. SampleRate=NumChannels*ScanRate. Has no effect when using and external clock.

STREAM_NUM_ADDRESSES
The number of entries in the scanlist

STREAM_SAMPLES_PER_PACKET
Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

STREAM_SETTLING_US
Time in microseconds to allow signals to settle after switching the mux. Default=0 equates to 10 us if sample rate <80k, and 6 us for higher sample rates. Max is 5000.

STREAM_RESOLUTION_INDEX
Index specifying the resolution of the data. High settings will have lower max speeds.

STREAM_BUFFER_SIZE_BYTES
Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2, as it takes 2 bytes to hold 1 sample. Changes while stream is running do not affect the currently running stream.

STREAM_CLOCK_SOURCE
Controls which clock source will be used to run the main stream clock. 0 = Internal crystal, 2 = External clock Source. T7 can accept an external clock on CIO3. All other values reserved.

STREAM_AUTO_TARGET
Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

STREAM_NUM_SCANS
The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Max number of samples is STREAM_BUFFER_SIZE_BYTES/2. Max number of scans is (STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES.

STREAM_EXTERNAL_CLOCK_DIVISOR
The number of pulses per scan when using an external clock.

STREAM_ENABLE
Write 1 to start stream. Write 0 to stop stream.

STREAM_SCANLIST_ADDRESS#(0:127)
A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Names	Addresses
STREAM_SCANLIST_ADDRESS0, STREAM_SCANLIST_ADDRESS1, STREAM_SCANLIST_ADDRESS2, Show All	4100, 4102, 4104, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Additional configuration notes

Additionally, address 4018 (STREAM_DATATYPE) must be written with the value 0. Note that address 4018 (STREAM_DATATYPE) is not in ljm_constants.json and is not compatible with LJM_NameToAddress.

STREAM_ENABLE must be written last.

The following registers do not necessarily need to be written, as the defaults may be adequate:

- STREAM_SETTLING_US
- STREAM_RESOLUTION_INDEX
- STREAM_BUFFER_SIZE_BYTES
- STREAM_NUM_SCANS

Data Collection

Once stream has been initiated with STREAM_ENABLE, the T7 sends data to the target indicated by STREAM_AUTO_TARGET until STREAM_ENABLE is written with the value 0. Stream-out streams that do not contain stream-in channels (see above) do not send data.

Modbus Feedback Auto-Response Packet Protocol

Bytes 0-1: Transaction ID

Bytes 2-3: Protocol ID

Bytes 4-5: Length, MSB-LSB

Bytes 6: 1 (Unit ID)

Byte 7: 76 (Function #)

Byte 8: 16

Byte 9: Reserved

Bytes 10-11: Backlog

Bytes 12-13: Status Code

Byte 14-15: Additional status information

Byte 16+: Stream Data (raw sample = 2 bytes MSB-LSB).

Status Codes:

2940: Auto Recovery Active.

2941: Auto Recovery End. Additional Status Information is the number of scans skipped.

2942: Scan Overlap

2943: Auto Recovery End Overflow

2944: Stream Burst Complete

Stop

To stop stream, write 0 to STREAM_ENABLE. All stream modes expect to be stopped, except for burst stream (see STREAM_NUM_SCANS for more information on bust stream).

Code Example

A general low-level stream example written in C/C++ can be found here.

3.2.2 Low-Level Streaming

4.0 Hardware Overview

The T7 has 3 different I/O areas:

- **Communication Edge:** The T7 has a USB type B connector and an RJ45 Ethernet connector. The T7-Pro has those and also has an SMA-RP female connector and a WiFi antenna. Power is always provided through the USB connector, even if USB communication is not used.
- **Screw Terminal Edge:** The screw terminal edge has convenient connections for 4 analog inputs, both analog outputs, 4 digital I/O, and both current sources. The screw terminals are arranged in blocks of 4, with each block consisting of VS, GND, and two I/O. Also on this edge are two LEDs. The Comm LED generally blinks with communication traffic, while the Status LED is used for other indications.
- **DB Edge:** The DB Edge has 2 D-sub type connectors: a DB15 and DB37. The DB15 has 12 digital I/O. The DB37 has the same I/O as the screw-terminals, plus additional analog inputs and digital I/O, for a total of 14 analog inputs, 2 analog outputs, 2 fixed current sources, and 11 digital I/O.

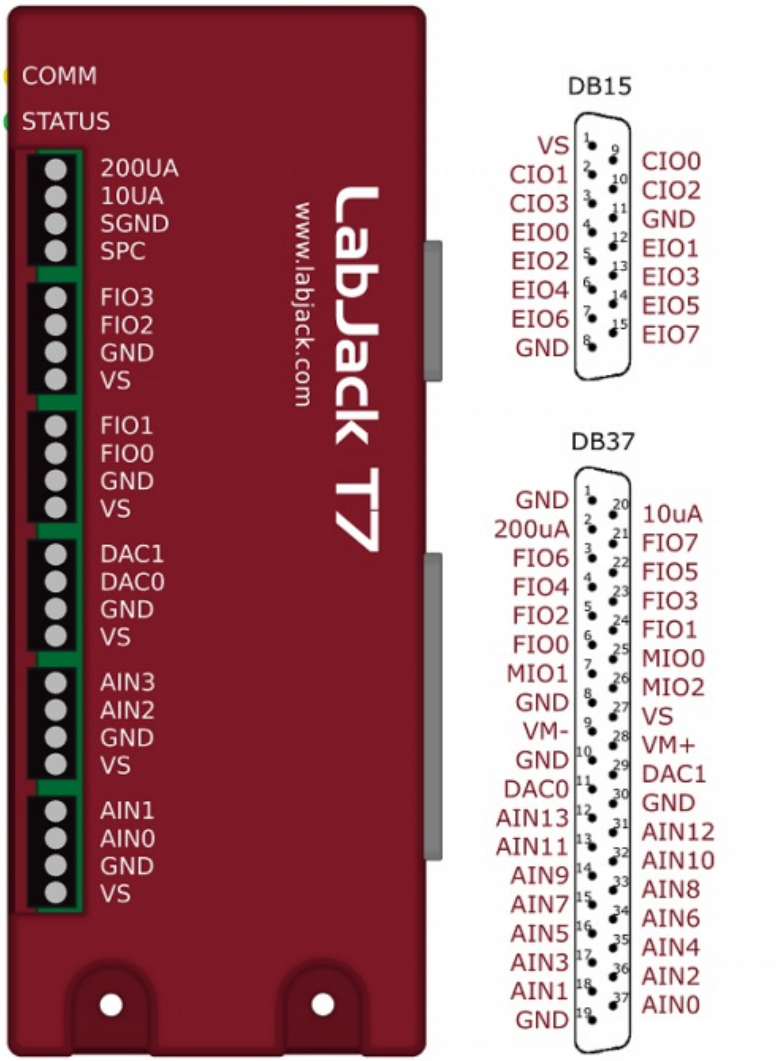


Figure 4-1. Enclosure & Connectors

USB: Can be used for communication. Power is always provided through this connector.

Ethernet: 10/100Base-T Ethernet connection can be used for communication.

WiFi (T7-Pro only): 2.4 GHz 802.11 b/g WiFi connection can be use for communication.

LEDs: The Power and Status LEDs convey different information about the device.

VS: All VS terminals are the same. These are outputs that can be used to source about 5 volts.

GND/SGND: All GND terminals are the same. SGND has a self-resetting thermal fuse in series with GND.

10UA/200UA: Fixed current sources providing 10µA/200µA at a max voltage of about 3 volts.

FIO#/EIO#/CIO#/MIO#: These are the 23 digital I/O, and are also referred to as DIO0-DIO22. Besides basic digital I/O operations, some of these terminals can also be configured with Extended Features (frequency input, pwm output, etc.), and all can be configured for various serial protocols: I2C serial, SPI serial, SBUS serial (EI-1050, SHT sensors), 1-Wire serial, and Asynchronous serial.

AIN#: AIN0-AIN13 are the 14 analog inputs.

DAC#: DAC0 & DAC1 are the 2 analog outputs. Each DAC can be set to a voltage between about 0.01 and 5 volts with 12-bits of resolution.

For information about reading inputs, start in Section 3. For information about setting outputs, start with the Waveform Generation Application Note.

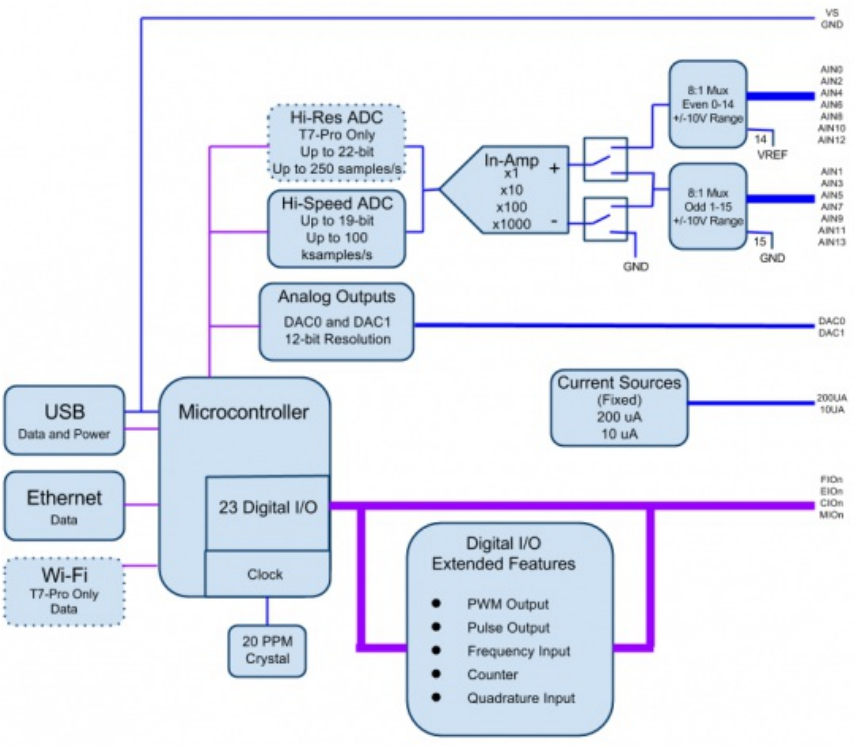


Figure 4-2. Block Diagram

General Device Information Registers

Name	Start Address	Type	Access	Default
PRODUCT_ID	60000	FLOAT32	R	
HARDWARE_VERSION	60002	FLOAT32	R	
FIRMWARE_VERSION	60004	FLOAT32	R	
BOOTLOADER_VERSION	60006	FLOAT32	R	
WIFI_VERSION	60008	FLOAT32	R	
HARDWARE_INSTALLED	60010	UINT32	R	0
ETHERNET_MAC	60020	UINT64	R	
WIFI_MAC	60024	UINT64	R	
SERIAL_NUMBER	60028	UINT32	R	
DEVICE_NAME_DEFAULT	60500	STRING	R/W	

PRODUCT_ID
The numeric identifier of the device. Such as 3 for a U3-HV.

HARDWARE_VERSION
The hardware version of the device.

FIRMWARE_VERSION
The current firmware version installed on the main processor.

BOOTLOADER_VERSION
The bootloader version installed on the main processor.

WIFI_VERSION
The current firmware version of the WiFi module, if available.

HARDWARE_INSTALLED
Bitmask indicating installed hardware options. bit0: High Resolution ADC, bit1: WiFi, bit2: RTC, bit3: uSD.

ETHERNET_MAC
The MAC address of the wired Ethernet module.

WIFI_MAC
The MAC address of the WiFi module.

SERIAL_NUMBER
The serial number of the device.

DEVICE_NAME_DEFAULT
The current device name. Up to 49 characters, cannot contain periods.

There are several other useful device control registers listed below. These registers have device-wide impact.

Miscellaneous Device Registers

Name	Start Address	Type	Access	Default
CORE_TIMER	61520	UINT32	R	
SYSTEM_REBOOT	61998	UINT32	W	
WAIT_US_BLOCKING	61590	UINT32	R/W	

CORE_TIMER
Internal 32-bit system timer running at 1/2 core speed, thus normally 80M/2 => 40 MHz.

SYSTEM_REBOOT
Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot.

WAIT_US_BLOCKING
Delays for x microseconds. Range is 0-100000.

5.0 USB

Communication Protocol: **Modbus TCP**

Connector Type: **USB-B Receptacle**

Compatible: **USB 1.1+**

Max Cable Length: **5 meters**

Max Packet Size: **64 bytes/packet**

Power is supplied to the T7 through the 5V USB connection. If the Ethernet or Wi-Fi connection is preferred for communication, use the provided AC USB 5V adapter for power. When used for communication, it is a full-speed USB connection compatible with USB version 1.1 or higher.

Interface - Talk to the T7

Modbus TCP is the protocol used by all connections on the T7 (USB, Ethernet, WiFi). If you want to handle USB communication yourself (find/open/write/read/close), you can use the Modbus protocol. Most customers, however, will use our LJM library which provides convenient device discovery, high-level functions, and programming flexibility.

Power Considerations

USB ground is connected to the T7 ground (GND), and USB ground is generally the same as the ground of the PC chassis and AC mains, since standard USB is non-isolated.

It is possible to isolate USB, and thereby protect the T7 from a power surge coming through the computer, if you use a USB isolator. USB isolators typically go for $40 to $100 USD, depending on the capabilities.

The T7-Pro will generally require a powered USB hub when in operating at full-power, since some USB ports/hubs will not supply the current necessary (500mA). Our experience with cheap USB supplies has shown them to be unreliable above 200mA. We recommend a powered USB hub rated for battery charging

applications, since these are typically rated for 1-2A. See electrical specifications for details on USB current requirements.

If designing your own driver...

The LabJack vendor ID is 0x0CD5. The product ID for the T7 is 0x0007.

The USB interface consists of the normal bidirectional control endpoint (0 OUT & IN), 3 used bulk endpoints (1 OUT, 2 IN, 3 IN), and 1 dummy endpoint (3 OUT). Endpoint 1 consists of a 64 byte OUT endpoint (address = 0x01). Endpoint 2 consists of a 64 byte IN endpoint (address = 0x82). Endpoint 3 consists of a dummy OUT endpoint (address = 0x03) and a 64 byte IN endpoint (address = 0x83). Endpoint 3 OUT is not supported by the firmware, and should never be used.

All commands should always be sent on Endpoint 1, and the responses to commands will always be on Endpoint 2. Endpoint 3 is only used to send stream data from the T7 to the host.

6.0 Ethernet

Communication Protocol: **Modbus TCP**

Connector Type: **RJ-45 Socket, Cat 5**

Compatible: **10/100Base-T**

PoE Compatible: **No**[1]

Max Cable Length: **100 meters typical**

Max Packet Size: **1040 bytes/packet (TCP), 64 bytes/packet (UDP)**

Overview

The T7 has a 10/100Base-T Ethernet connection. This connection only provides communication, so power must be provided through the USB connector. Refer to this WiFi and Ethernet tutorial to get started.

Note: If you need a wireless connection instead of Ethernet, you can buy a wireless bridge, and connect the T7 to the bridge, and the bridge will connect to the wireless network. Find more information in the Convert Ethernet to WiFi App Note.

Config (_DEFAULT) Registers versus Status Registers

The first list below is the config registers. These are generally written to configure default Ethernet settings, but can also be read. Configure the Ethernet parameters in Kipling software.

Any register with _DEFAULT at the end is non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle.

The second list below consists of read-only registers to read the status of Ethernet.

For example, if you write ETHERNET_IP_DEFAULT="192.168.1.207" (you actually write/read the 32-bit numeric equivalent not an IP string), then that value will be retained through reboots and is the default IP address. If DHCP is disabled, this will be the static IP of the device and what you get if you read ETHERNET_IP. If DHCP is enabled, then a read of ETHERNET_IP will return the IP set by the DHCP server.

Ethernet Config Registers

Name	Start Address	Type	Access	Default
ETHERNET_IP_DEFAULT	49150	UINT32	R/W	
ETHERNET_SUBNET_DEFAULT	49152	UINT32	R/W	
ETHERNET_GATEWAY_DEFAULT	49154	UINT32	R/W	
ETHERNET_DNS_DEFAULT	49156	UINT32	R/W	
ETHERNET_ALTDNS_DEFAULT	49158	UINT32	R/W	
ETHERNET_DHCP_ENABLE_DEFAULT	49160	UINT16	R/W	

ETHERNET_IP_DEFAULT
The IP address of wired Ethernet after a power-cycle to the device.

ETHERNET_SUBNET_DEFAULT
The subnet of wired Ethernet after a power-cycle to the device.

ETHERNET_GATEWAY_DEFAULT
The gateway of wired Ethernet after a power-cycle to the device.

ETHERNET_DNS_DEFAULT
The DNS of wired Ethernet after a power-cycle to the device.

ETHERNET_ALTDNS_DEFAULT
The Alt DNS of wired Ethernet after a power-cycle to the device.

ETHERNET_DHCP_ENABLE_DEFAULT
The Enabled/Disabled state of Ethernet DHCP after a power-cycle to the device.

Ethernet Status Registers

Name	Start Address	Type	Access	Default
ETHERNET_IP	49100	UINT32	R	
ETHERNET_SUBNET	49102	UINT32	R	
ETHERNET_GATEWAY	49104	UINT32	R	
ETHERNET_DNS	49106	UINT32	R	
ETHERNET_ALTDNS	49108	UINT32	R	
ETHERNET_DHCP_ENABLE	49110	UINT16	R	

ETHERNET_IP
Read the current IP address of wired Ethernet.

ETHERNET_SUBNET
Read the current subnet of wired Ethernet.

ETHERNET_GATEWAY
Read the current gateway of wired Ethernet.

ETHERNET_DNS
Read the current DNS of wired Ethernet.

ETHERNET_ALTDNS
Read the current Alt DNS of wired Ethernet.

ETHERNET_DHCP_ENABLE
Read the current Enabled/Disabled state of Ethernet DHCP.

Some Examples

Read IP Example: To read the wired IP Address of a device, perform a modbus read of address 49100. The value will be returned as an unsigned 32-bit number, such as 3232235691. Change this number to an IP address by converting each binary group to an octet, and adding decimal points as necessary. The result in this case would be "192.168.0.171".

Change IP Example: To change the Ethernet IP Address of a device, perform a modbus write to address 49150. The value must be passed as an unsigned 32-bit number, such as 3232235691. Change this IP address "192.168.0.171" by converting each octet to a binary group, and sticking them together.

More Details

Once default Ethernet configuration register(s) are changed, the current settings will be updated on the next power cycle. Alternatively, toggle power to the Ethernet module by writing a 0, then a 1 to the **POWER_ETHERNET** address.

Ethernet Power Settings

Name	Start Address	Type	Access	Default
POWER_ETHERNET	48003	UINT16	R/W	
POWER_ETHERNET_DEFAULT	48053	UINT16	R/W	

POWER_ETHERNET
The current ON/OFF state of the Ethernet module. Provided to optionally reduce power consumption.

POWER_ETHERNET_DEFAULT
The ON/OFF state of the Ethernet module after a power-cycle to the device.

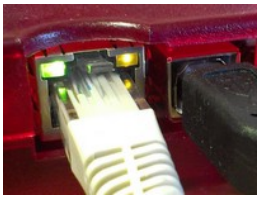
Isolation

The Ethernet connection on the T7 has 1500 volts of galvanic isolation. All power supplies shipped by LabJack Corporation with the T7 have at least 500 volts of isolation.

Note that if you power the T7 from a USB host/hub, ground from the host/hub is typically connected to upstream USB ground which often finds its way back to AC mains ground and thus there would be no isolation.

LEDs on the Ethernet jack

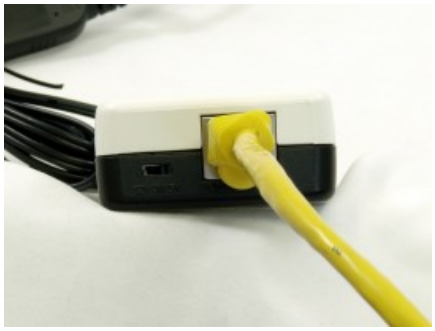
Both the green and orange LEDs on the Ethernet jack will illuminate on connection to an active Ethernet cable. The orange LED turns on when an active link is detected and blinks when packets are received/processed. The green LED illuminates when the connection is 100Mbps.



The orange LED is closest to the USB connector.

[1] The T7 cannot be directly powered via a Power over Ethernet (PoE) cable. However, it is relatively easy to find a PoE splitter that converts 48V on PoE to the 5V necessary for the T7. Such adapters run from ~$30 to ~$50 USD. Used in combination, the following parts work to split PoE:

- TP-LINK TL-POE10R - To split 48V from the Ethernet cable into a 5.5mm OD, 2.1mm ID center positive barrel receptacle.
- Tensility International Corp 10-00240 - To convert 5.5mm OD, 2.1mm ID center positive barrel connector to USB-A male plug.
- Tensility International Corp 10-00648 - A Female to B Male USB cable. This will fit on the USB-A male plug (Tensility 10-00240), and insert into the T7.



7.0 WiFi (T7-Pro only)

Communication Protocol: **Modbus TCP**

Connector Type: **Female RP-SMA**

Transceiver: **2.4 GHz 802.11 b/g** (Compatible with ac, and n routers)

Range: **With stock antenna ... similar to laptops and other WiFi devices**

Max Packet Size: **500 bytes/packet**

Overview

The T7-Pro has a wireless module. Refer to this WiFi and Ethernet tutorial to get started.

DHCP is enabled from the factory, so to get WiFi going from the factory write the desired SSID string (case sensitive) to WIFI_SSID_DEFAULT and the proper password string (case sensitive) to WIFI_PASSWORD_DEFAULT. Then write a 1 to WIFI_APPLY_SETTINGS and watch the status codes. If you get back code 2900 the WiFi chip is associated to your network, and you can then read the assigned IP from WIFI_IP. Find more details and troubleshooting tips in the WiFi and Ethernet tutorial.

Note: It's possible to get data faster on a T7-Pro using its Ethernet interface instead of its WiFi interface, both for command response and streaming modes. Wireless bridges and access points do not introduce a speed bottleneck, so a bridge is a good way to get fast wireless data from a T7, or T7-Pro. See the Convert Ethernet to WiFi App Note for setup information.

Config (_DEFAULT) Registers versus Status Registers

The first list below is the config registers. These are generally written to configure default WiFi settings, but can also be read (except password). Configure the WiFi parameters in Kipling software.

Any register with _DEFAULT at the end is non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle. WiFi is unique compared to other systems on the T7, in that new values written to _DEFAULT registers are not actually saved until a 1 is written to WIFI_APPLY_SETTINGS.

The second list below consists of read-only registers to read the status of WiFi.

For example, if you write WIFI_IP_DEFAULT="192.168.1.208" (you actually write/read the 32-bit numeric equivalent not an IP string), then that value will be retained through reboots and is the default IP address. If DHCP is disabled, this will be the static IP of the device and what you get if you read WIFI_IP. If DHCP is enabled, then a read of WIFI_IP will return the IP set by the DHCP server.

WiFi Config Registers

Name	Start Address	Type	Access	Default
WIFI_IP_DEFAULT	49250	UINT32	R/W	
WIFI_SUBNET_DEFAULT	49252	UINT32	R/W	
WIFI_GATEWAY_DEFAULT	49254	UINT32	R/W	
WIFI_DHCP_ENABLE_DEFAULT	49260	UINT16	R/W	
WIFI_SSID_DEFAULT	49325	STRING	R/W	
WIFI_PASSWORD_DEFAULT	49350	STRING	W	
WIFI_APPLY_SETTINGS	49400	UINT32	W	

WIFI_IP_DEFAULT
The new IP address of WiFi. Use WIFI_APPLY_SETTINGS.

WIFI_SUBNET_DEFAULT
The new subnet of WiFi. Use WIFI_APPLY_SETTINGS.

WIFI_GATEWAY_DEFAULT
The new gateway of WiFi. Use WIFI_APPLY_SETTINGS.

WIFI_DHCP_ENABLE_DEFAULT
The new Enabled/Disabled state of WiFi DHCP. Use WIFI_APPLY_SETTINGS

WIFI_SSID_DEFAULT
The new SSID (network name) of WiFi. Use WIFI_APPLY_SETTINGS.

WIFI_PASSWORD_DEFAULT
Write the password for the WiFi network, then use WIFI_APPLY_SETTINGS.

WIFI_APPLY_SETTINGS
Apply all new WiFi settings: IP, Subnet, Gateway, DHCP, SSID, Password. 1=Apply

WiFi Status Registers

Name	Start Address	Type	Access	Default
WIFI_IP	49200	UINT32	R	
WIFI_SUBNET	49202	UINT32	R	
WIFI_GATEWAY	49204	UINT32	R	
WIFI_DHCP_ENABLE	49210	UINT16	R	
WIFI_SSID	49300	STRING	R	
WIFI_STATUS	49450	UINT32	R	
WIFI_RSSI	49452	FLOAT32	R	

WIFI_IP

Read the current IP address of WiFi.

WIFI_SUBNET
Read the current subnet of WiFi.

WIFI_GATEWAY
Read the current gateway of WiFi.

WIFI_DHCP_ENABLE
Read the current Enabled/Disabled state of WiFi DHCP.

WIFI_SSID
Read the current SSID (network name) of WiFi

WIFI_STATUS
Status Codes: ASSOCIATED = 2900, ASSOCIATING = 2901, ASSOCIATION_FAILED = 2902, UNPOWERED = 2903, BOOTING = 2904, START_FAILED = 2905, APPLYING_SETTINGS = 2906, DHCP_STARTED = 2907, OTHER = 2909

WIFI_RSSI
WiFi RSSI (signal strength). Typical values are -40 for very good, and -75 for very weak. The T7 microcontroller only gets a new RSSI value from the WiFi module when WiFi communication occurs.

WiFi Power Registers

Name	Start Address	Type	Access	Default
POWER_WIFI	48004	UINT16	R/W	
POWER_WIFI_DEFAULT	48054	UINT16	R/W	

POWER_WIFI
The current ON/OFF state of the WiFi module. Provided to optionally reduce power consumption.

POWER_WIFI_DEFAULT
The ON/OFF state of the WiFi module after a power-cycle to the device.

Examples

Read IP Example: To read the wireless IP address of a device, perform a modbus read of address 49200. The value will be returned as an unsigned 32-bit number, such as 3232235691. Change this number to an IP address by converting each binary group to an octet, and adding decimal points as necessary. The result in this case would be "192.168.0.171"

Write IP Example: To change the Wireless IP Address of a device, perform a modbus write to address 49250. The IP address must be passed as an unsigned 32-bit number, such as 3232235691. Change this IP address "192.168.0.171" by converting each octet to a binary group, and sticking them together.

More Details

Once default wireless configuration register(s) are changed, it is necessary to also write 1 to the **WIFI_APPLY_SETTINGS** register. Alternatively, the default settings will be updated on the next power cycle.

Update WiFi Firmware

The WiFi chip on the T7 is a separate chip from the main processor, and it can be updated using the WIFI_FIRMWARE_UPDATE_TO_VERSIONX register. If connected to the internet, the WiFi chip can download new firmware files from an ftp server. To initiate a download and update, write a new firmware version to the WIFI_FIRMWARE_UPDATE_TO_VERSIONX register. *At the time of this writing we recommend using Kipling to update WiFi firmware, since Kipling connects to the FTP server to identify what firmware is available, and monitors the WIFI_FIRMWARE_UPDATE_STATUS register automatically.*

WiFi Firmware Registers

Name	Start Address	Type	Access	Default
WIFI_VERSION	60008	FLOAT32	R	
WIFI_FIRMWARE_UPDATE_TO_VERSIONX	49402	FLOAT32	W	

WIFI_FIRMWARE_UPDATE_STATUS	49454	UINT32	R

WIFI_VERSION
The current firmware version of the WiFi module, if available.

WIFI_FIRMWARE_UPDATE_TO_VERSIONX
Start an update by using USB or Ethernet to write the desired version to update to.

WIFI_FIRMWARE_UPDATE_STATUS
CONFIGURING = 2920, IN_PROGRESS = 2921, REBOOTING = 2923, UPDATE_SUCCESS = 2924, UPDATE_FAILED = 2925

Antenna Details

The T7-Pro ships with a standard RP-SMA 2.4 GHz antenna similar to the W1030, or the A24-HASM-450.

The connection to the WiFi module on the T7-Pro PCB is made via a snap-on/snap-off ultra miniature coaxial connector called male U.FL (aka AMC, IPEX, IPAX, IPX, MHF, UMC or UMCC). The normal T7-Pro uses a U.FL to bulkhead RP-SMA cable with a length of 140mm, similar to the Taoglas Limited CAB.622, Emerson 415-0100-150, Laird 1300-00041, Amphenol 336306-14-0150, or Amphenol 336306-12-0150.

To search for U.FL to RP-SMA cable options at Digikey, go to the "Cable Assemblies => Coaxial Cables (RF)" section, and filter by Style = "RP-SMA to IPX" or "RP-SMA to MHF1" or "RP-SMA to UMC" or "RP-SMA to UMCC". Then look at the picture and make sure it looks correct, as the application of the terms male and female are not totally standardized.

T7-Pro-OEM ships with a simple 30mm U.FL whip antenna such as the Anaren 66089-2406.

WiFi Range

The WiFi range on the T7 is typical for a modern WiFi device. In direct line-of-sight with the router, it's possible to get a decent connection at 100m. The table below shows signal strength at varying distances with a stock T7 antenna, and a simple WiFi router. Both the T7 and the router were positioned 3ft off of the ground, with direct line-of-sight.

Distance	RSSI
10m	-44dBm
25m	-45dBm
50m	-55dBm
100m	-59dBm

During testing, it was noted that the T7 had slightly better WiFi range than an HTC One V cell phone. The WiFi signal is spotty at RSSI lower than -75dBm, and the connection will cut off entirely around -80dBm. Note that 90° antenna orientation was used in testing above. That is to say, keep the antenna in the fully bent upright position, don't try to point it at the router, or accidentally leave it at 45° bent. At 45° bent, or directly pointed towards the router, the signal strength is reduced by about 5dBm.

Note that the RSSI value you can read (WIFI_RSSI) is only updated when WiFi communication occurs. That is, if you talk to the T7 over USB to read the RSSI value, you will just get the value from the last WiFi communication that occurred.

OEM Whip Antenna

The OEM whip antenna is a short segment of wire, only 30mm in length. This whip antenna provides an inexpensive solution for adding WiFi to an OEM board, without the need to figure out mounting of a bigger antenna. The signal strength of a 30mm whip antenna is on average 11dB less than that of the stock antenna. The table below demonstrates the 30mm antenna signal strength at various distances.

Distance	RSSI
10m	-49dBm
25m	-58dBm
50m	-70dBm
100m	-73dBm

Improve signal strength

The first step to improve the signal strength at large distances is to insure direct line-of-sight. Beyond that, the next best thing is to elevate the transmitter and receiver antennas. Even an elevation of 1m off the ground helps quite a bit. Be sure to consider the probability of lightning strikes if the antenna is high relative to the surroundings.

The next step to improve signal strength is to use a directional WiFi antenna. Directional antennas improve range substantially, such that even a homemade solution can increase range to fifteen times that of a non-directional antenna. If you need something to work at 500m, it's possible to buy a simple yagi antenna for $60 USD approx.

Network Diagrams

Figure 1 demonstrates a basic network diagram where the T7-Pro is connected to the "Office WiFi" network. The T7-Pro can be controlled by either host option (1 or 2). For more information on how to setup a WiFi network, and some basic troubleshooting steps, see the Basic Networking & Troubleshooting App Note.

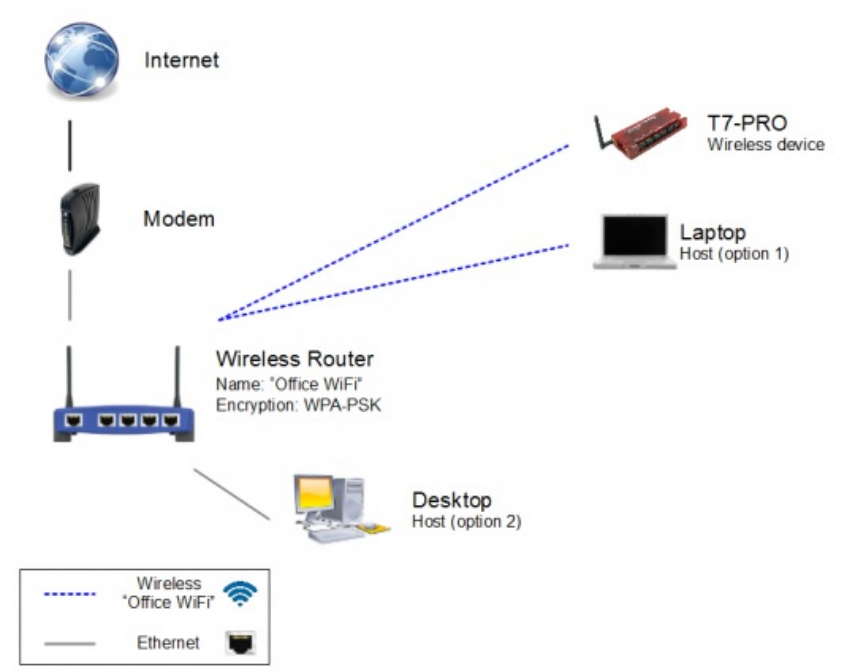


Figure 1. Most common configuration for a home, or small office network.

In a slightly larger network, there may be more computers, and several wireless networks (wireless access points, APs). Figure 2 demonstrates a medium sized network where there are two APs, the Wireless Router creates one, and the Wireless Access Point creates another. The Wireless Access Point might be on the manufacturing floor, outdoors, in the lab, etc. Note that it's possible to control the T7s using any of the host computer options (1-3).

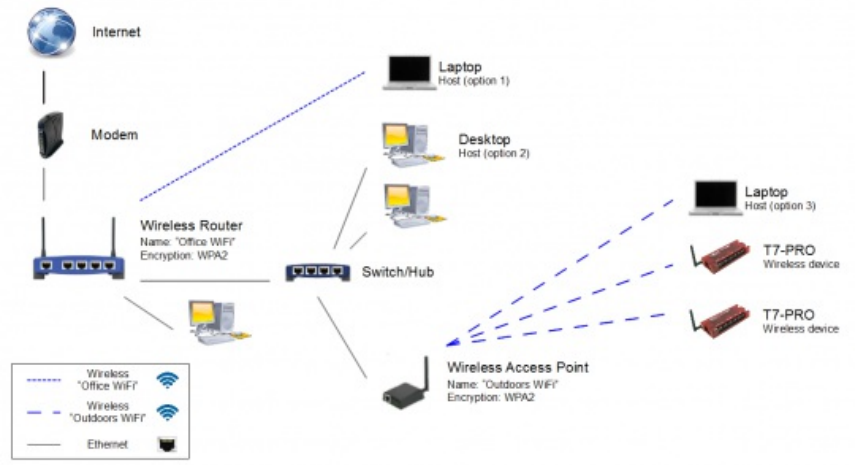


Figure 2. A medium sized office network, with a secondary wireless network (AP) for other devices.

It's possible to communicate with T7-Pros using only a laptop and WiFi, as shown in figure 3, and figure 4. Simply configure the laptop to create a wireless network (wireless access point, AP). Windows 7 and other new operating systems can configure their wireless cards to become an AP, or do an Internet search to find a simple program that sets up the virtual access point for you (we have found that My WiFi Router works well). The laptop can control the T7 (act as host), or simply provide a link for the T7-Pro to access other parts of a network.

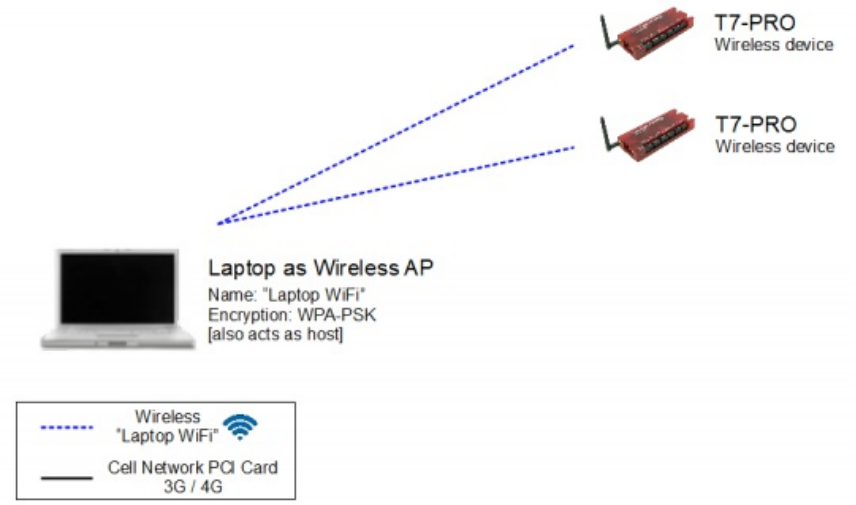


Figure 3. Laptop only. Laptop software and built-in wireless card create a wireless network (AP).

If you need internet access on your laptop, but you are using it as a wireless access point, then consider a 3G card, or USB dongle to get a cellular internet connection. Alternatively, locate an Ethernet jack that you know is connected to the internet.

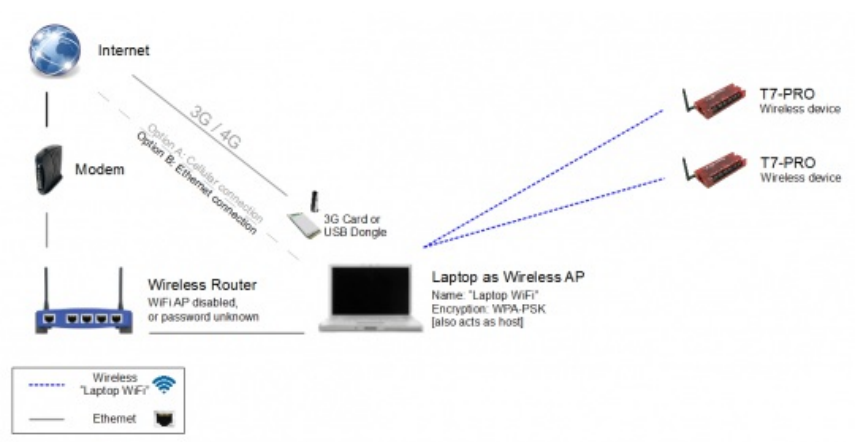


Figure 4. Laptop only (with internet). Either Option A or Option B can be used to get internet access. Just like figure 3, the laptop is creating the wireless network (AP).

If you have a cell phone(or tablet) and a laptop, you can create a Mobile Hotspot with the cell phone, and then connect the T7s and your laptop to the hotspot. Figure 5 demonstrates how to use a Mobile Hotspot with a laptop and T7. The Laptop acts as the host for the T7s, and can get internet through the cell phone's cellular network connection.

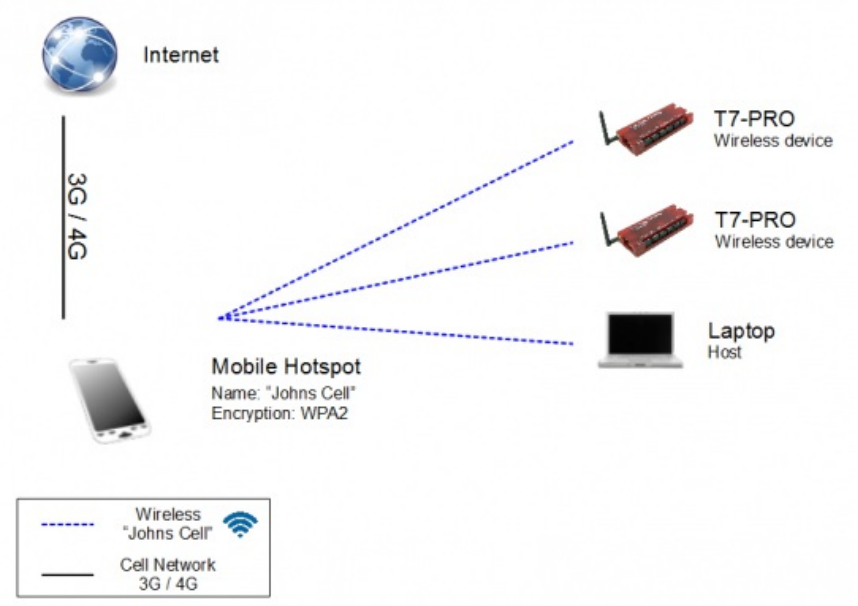


Figure 5. Laptop and Mobile Hotspot. The cell phone creates the wireless network (AP).

Note that 3G and 4G depend on your cell provider. Unlimited data plans often throttle your connection after the data limit is exceeded. Due to the data limits on cell networks, we advise caution for internet users on these setups. Figure 3 is still a good option to create a wireless local network, but the data will usually stay on the host computer.

7.0 WiFi

8.0 LEDs

STATUS - green LED

The status LED indicates when the T7 is performing autonomous tasks, such as running a Lua script or streaming. The LED will blink when a script accesses a modbus register or when a stream packet is prepared.

The status LED also activates during firmware updates to indicate various stages of the process, refer to the Combined LED Activity section.

COMM - yellow LED

The primary indicator for packet transfer. If the T7 is communicating the COMM LED will be blinking. A few blinks after connecting to the PC indicates that the T7 is enumerating. Enumeration is when the standard USB initialization takes place, and the host is gathering device information.

The COMM LED will blink when the T7 receives Modbus commands, or when streaming data. Each packet will produce a single blink. If commands are issued rapidly, the LED will blink rapidly. At high packet transfer rates the LED will blink at 10Hz, even though more than 10 packets are being processed per second.

Normal Power-Up Behavior

1. Both LEDs blink rapidly for about 1 second.
2. COMM solid off and STATUS solid on.
3. If USB enumerates, COMM blinks a few times and then stays solid on.

Combined LED Activity

When the LEDs blink together, the T7 is computing checksums.

When the LEDs are alternating, the T7 is copying a firmware image.

LED Config Registers

Name	Start Address	Type	Access	Default
POWER_LED	48006	UINT16	R/W	

POWER_LED_DEFAULT	48056	UINT16	R/W

POWER_LED
Sets the LED operation: 0 = Off. Useful for lower power applications. 1 = normal. 2 = Lower power, LEDs will still blink but will normally be off. 3 = Reserved. 4 = Manual, in this mode the LEDs can be user controlled.

POWER_LED_DEFAULT
The ON/OFF state of the LEDs after a power-cycle to the device.

9.0 VS, Power Supply

Supply Voltage: **4.75 - 5.25 volts (5V ±5% Regulated)**

Typical Active Current: **300 mA**

Typical Sleep Current: **8 mA**

Normal Power Connector: **USB-B Receptacle**

Typical Power Supply: **Any USB-Style Supply**

VS Terminals

The VS terminals are designed as outputs for the supply voltage. The supply voltage is nominally 5 volts and typically provided through the USB connector.

All VS terminals are the same.

The VS connections are outputs, not inputs. Do not connect a power source to VS in normal situations.

The max total current that can be drawn from VS is 500mA - DeviceSupplyCurrent, so if the T7 needs 250mA to run, that leaves 250mA available from the VS terminals.

The voltage on VS can be noisy and can change unexpectedly. Circuits that are sensitive to changing or noisy supply voltage, such as bridge circuits, should not be supplied from VS.

Power Supply

Power supply for the T7 is typically provided through the USB connector. For a different board-level connection option see "Alternate Power Supply" in the OEM section. Typical power supply sources include:

- USB host or hub.
- Wall-wart power supply with USB connection (included with normal retail units ... not OEM).
- Power-over-Ethernet splitter (e.g. TP-Link TL-POE10R with Tensility 10-00240 with Tensility 10-00648).
- Car charger with USB ports (e.g. Anker 71AN2452C-WA).
- Rechargeable battery with USB ports (e.g. Anker Astro E5 79AN15K-BA perhaps with Belkin F3U133-06INCH).
- Battery with car charger (e.g. Anker 79AN15K-BA with 71AN2452C-02WA).
- Battery with solar panel (e.g. Anker 79AN15K-BA with 71ANSCP-B145A).
- Pigtail a cable with a USB-B connector to get at the red & black wires, and make some sort of custom cable for your 5V power supply.

The supply range for specified operation is 4.75 to 5.25 volts, which is the same as the USB specification for voltage provided to a device. Nonetheless, we have seen some USB host ports providing a lower voltage. If your USB host port has this problem, add a USB hub with a strong power supply.

See information about Power over Ethernet (PoE) at the end of the Ethernet Section.

See related data in the General section of Specifications.

Normal retail units (not OEM) include a 5V, 2A wall-wart style power supply:

Compatibility	Make	Mfr. Model No.
North America	VA-PSU-US1	JX-B0520B-1-B

Europe	VA-PSU-EU1	JX-B0520A-1-B
United Kingdom	VA-PSU-UK1	JX-B0520C-1-B
Australia	-	JX-B0520D-1-B
China	-	JX-B0520H-1-B

Note that the JX-B0520 supply is rated for 0 to 40 deg C operation.

10.0 SGND and GND

SGND

SGND is located near the upper-left of the device. This terminal has a self-resetting thermal fuse in series with GND. This is often a good terminal to use when connecting the ground from another separately powered system that could unknowingly already share a common ground with the T7.

See the AIN, DAC, and Digital I/O(FIO, EIO, CIO, MIO) application notes for more information about grounding.

GND

The GND connections available at the screw-terminals and DB connectors provide a common ground for all LabJack functions. All GND terminals are the same and connect to the same ground plane.

GND is also connected to the ground pin on the USB connector, so if there is a connection to a USB port on a hub/host (as opposed to just a power supply connection), then GND is the same as the ground line on the USB connection, which is often the same as ground on the PC chassis, which is often the same as AC mains ground.

See the AIN, DAC, and Digital I/O (FIO, EIO, CIO, MIO) Sections for more information about grounding.

The max total current that can be sunk into GND is 500mA - DeviceSupplyCurrent, so if the T7 needs 250mA to run, the current sunk into GND terminals should be limited to 250mA. Note that sinking substantial current into GND can cause slight voltage differences between different ground terminals, which can cause noticeable errors with single-ended analog input readings.

11.0 SPC

SPC outputs diagnostic timing signals while streaming (see Stream Section for details), and is also used to force special startup behavior.

To force special startup behavior, securely install a short jumper wire from SPC to one of the following digital I/O lines:

- **FIO0**: Force boot to main firmware (internal) image. Used to boot the internal firmware even if its checksum is bad.
- **FIO1**: Force copy of backup image to overwrite internal image. Used to load the external firmware even if its checksum is bad.
- **FIO2**: Factory reset. Sets the start up configuration to factory settings. Disables Lua script at startup.
- **FIO3**: Load emergency image. This option loads a firmware image with minimal functionality (kinda like Windows safe-mode). Used to recover from firmware corruption or bugs. The update process is about all that can be done while in this mode.

12.0 200uA and 10uA

Overview

The T7 has 2 fixed current source terminals useful for measuring resistance (thermistors, RTDs, resistors). The 10UA terminal provides about 10 µA and the 200UA terminal provides about 200 µA, but the actual values should be read from the calibration constants, or better yet measured in real-time using a fixed shunt resistor.

Using the equation V=IR, with a known current and voltage, it is possible to calculate the resistance of the item in question. Figure 12-1 shows a simple setup measuring 1 resistor.

The factory value of each current source is noted during calibration and stored with the calibration constants on the device. These can be viewed using the Kipling software, or read programmatically. Note that these are fixed constants stored during calibration, not some sort of real-time readings.

Constant Current Sources

Name	Start Address	Type	Access	Default
CURRENT_SOURCE_200UA_CAL_VALUE	1902	FLOAT32	R	
CURRENT_SOURCE_10UA_CAL_VALUE	1900	FLOAT32	R	

CURRENT_SOURCE_200UA_CAL_VALUE

CURRENT_SOURCE_10UA_CAL_VALUE

For example: To read the factory value of the 200uA current source, perform a read of Modbus address 1902, and the result would be in the form of a floating point number, e.g. 0.000197456 amps.

Some Examples

Multiple resistances can be measured by putting them in series and measuring the voltage across each. Some applications might need to use differential inputs to measure the voltage across each resistor, but for many applications it works just as well to measure the single-ended voltage at the top of each resistor and subtract in software.

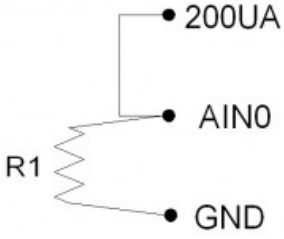


Figure 12-1

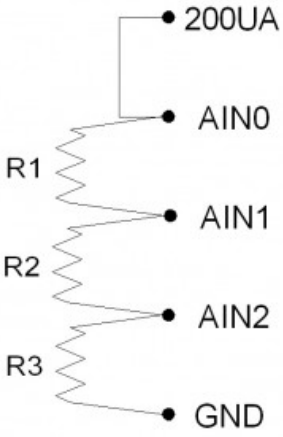


Figure 12-2

Figure 12-1 shows a simple setup measuring 1 resistor. If R1=3k, the voltage at AIN0 will be 0.6 volts.

Figure 12-2 shows a setup to measure 3 resistors using single-ended analog inputs. If R1=R2=R3=3k, the voltages at AIN0/AIN1/AIN2 will be 1.8/1.2/0.6 volts. That means AIN0 and AIN1 would be measured with the +/-10 volt range, while AIN2 could be measured with the +/-1 volt range. This points out a potential advantage to differential measurements, as the differential voltage across R1 and R2 could be measured with the +/-1 volt range, providing better resolution.

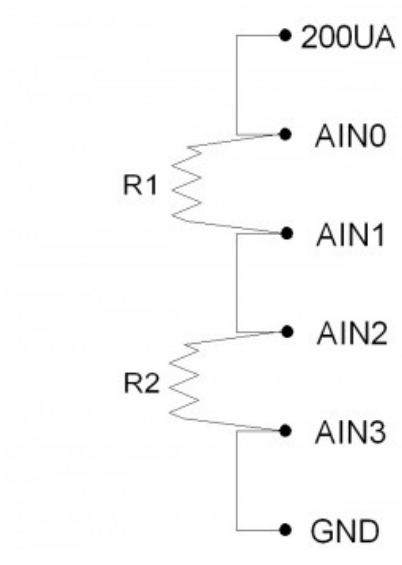


Figure 12-3

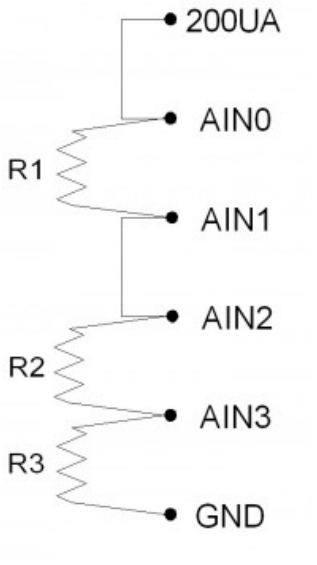


Figure 12-4

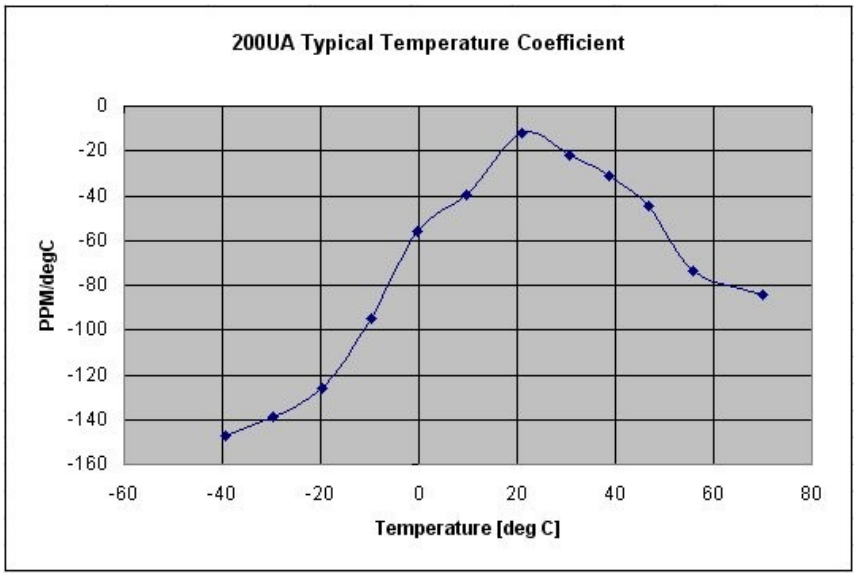
Figure 12-3 shows a setup to measure 2 resistors using differential analog inputs. AIN3 is wasted in this case, as it is connected to ground, so a differential measurement of AIN2-AIN3 is the same as a single-ended measurement of AIN2. That leads to Figure 12-4, which shows R1 and R2 measured differentially and R3 measured single-ended.

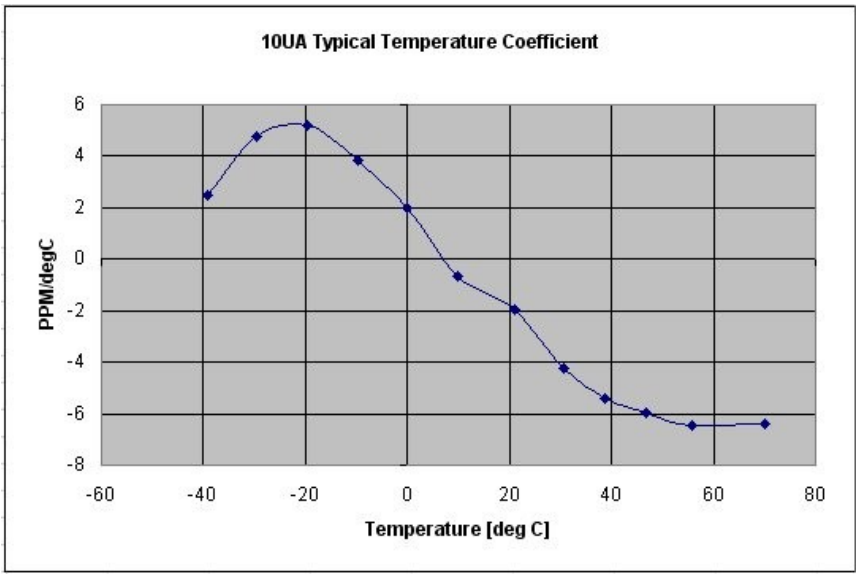
Specifications

The current sources can drive about 3 volts max, thus limiting the maximum load resistance to about 300 kΩ (10UA) and 15 kΩ (200UA). Keep in mind that high source resistance could cause settling issues for analog inputs.

The current sources have good accuracy and tempco, but for improvement a fixed resistor can be used as one of the resistors in the figures above. The Y1453-100 and Y1453-1.0K from Digikey have excellent accuracy and very low tempco. By measuring the voltage across one of these you can calculate the actual current at any time.

The following charts show the typical tempco of the current sources over temperature. The 10UA current source has a very low tempco across temperature. The 200 UA current source has a good tempco from about 0-50 degrees C, and outside of that range the effect of tempco will be more noticeable.





Example - PT100 or PT1000 RTD

Assume that R1 in Figure 12-1 is a PT100 RTD. A PT100 RTD is 100 ohms at 0 degC. The response of an RTD is nonlinear, but the linear slope 0.384 ohms/degC works well from about -40 to +150 degC. That leads to the following expression:

R = (0.384 * DegC) + 100

... which can be rearranged to:

DegC = (2.604 * R) - 260.4

We are determining R by measuring the voltage that results from a known current passed through R, that is R = V/I, so we can say:

DegC = (2.604 * V/I) - 260.4

This tells us that the slope is 2.604/I and the offset is -260.4. To determine I, you can just use 0.0002 amps, or use the factory calibration value read from CURRENT_SOURCE_200UA_CAL_VALUE, or use a precision fixed resistor as mentioned above to measure I in real time. Assume we read the factory cal value as 0.000200 amps, and thus use a constant slope of 2.604/0.0002 = 13020. We can now use the AIN-EF feature to apply this slope and offset:

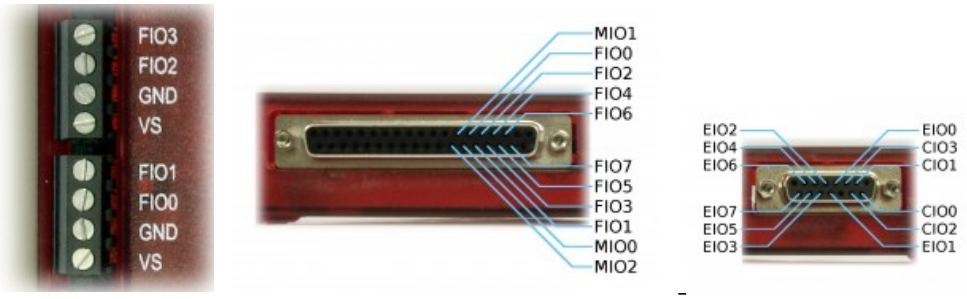
```
AIN0_EF_INDEX = 1              // feature index for type K thermocouple
AIN0_EF_CONFIG_D = 13020.0     // slope
AIN0_EF_CONFIG_E = -260.4      // offset
```

Now reads of AIN0_EF_READ_A will return (13020.0 * volts) - 260.4.

Note that you can come up with your own slope & offset for your temperature region of interest. For example, we made this Google Spreadsheet and decided that Slope=2.720 (degC/ohm) and Offset=-277.5 works best for the region of 100 to 300 degC.

Note that a PT1000 simply has 10x the response of a PT100 (~3.84 ohms/degC). The offset still works out to -260.4, but the slope is 0.260.

13.0 Digital I/O



Digital I/O: **23**

Logic Level: **3.3V**

DIO is a generic name used for all digital I/O. The DIO are subdivided into different ports called FIO, EIO, CIO, and MIO. For wiring information on open-collector signals, driven signals, controlling relays, and mechanical switches, see the digital signals app note.

	FIO (0-7)								EIO (0-7)								CIO (0-3)				MIO (0-2)		
DIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
DIO Mapping																							

There are 4 types of registers used for digital I/O interaction: Simple DIO#, DIO State, DIO Direction, and DIO Inhibit

Simple Digital I/O#

Read or set the state of 1 digital I/O. Automatically configures the direction to input (when reading) or to output (when writing).

```
FIO0-FIO7 = DIO0-DIO7
EIO0-EIO7 = DIO8-DIO15
CIO0-CIO3 = DIO16-DIO19
MIO0-MIO2 = DIO20-DIO22
```

Digital I/O

Name	Start Address	Type	Access	Default
FIO#(0:7)	2000	UINT16	R/W	
EIO#(0:7)	2008	UINT16	R/W	
CIO#(0:3)	2016	UINT16	R/W	
MIO#(0:2)	2020	UINT16	R/W	

FIO#(0:7)
Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
FIO0, FIO1, FIO2, Show All	2000, 2001, 2002, Show All

EIO#(0:7)
Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
EIO0, EIO1, EIO2, Show All	2008, 2009, 2010, Show All

CIO#(0:3)
Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
CIO0, CIO1, CIO2, Show All	2016, 2017, 2018, Show All

MIO#(0:2)
Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
MIO0, MIO1, MIO2	2020, 2021, 2022

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Digital I/O State Bit Masks

Each of these is a single binary-encoded value representing the state of 8 bits of I/O. Each bit represents an I/O line. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of these values are inhibits. The inhibit bits prevent the corresponding state bit from being modified.

State Bit Masks

Name	Start Address	Type	Access	Default
FIO_STATE	2500	UINT16	R/W	
EIO_STATE	2501	UINT16	R/W	
CIO_STATE	2502	UINT16	R/W	
MIO_STATE	2503	UINT16	R/W	

FIO_STATE
Read or write the state of the 8 bits of FIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

EIO_STATE
Read or write the state of the 8 bits of EIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

CIO_STATE
Read or write the state of the 4 bits of CIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

MIO_STATE
Read or write the state of the 3 bits of MIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

For example: To read the digital state of all FIO lines in a bit mask, read FIO_STATE. The value will be something like 0b11111011 representing 1 for logic high, and 0 for logic low. FIO2 is currently logic low.

Digital I/O Direction Bit Masks

Each of these is a single binary-encoded value representing the direction of 8 bits of I/O. Each bit designates an I/O line. 0=Input and 1=Output. The upper 8-bits of this value are inhibits. The inhibit bits prevent the corresponding direction bit from being modified.

Direction Bit Masks

Name	Start Address	Type	Access	Default
FIO_DIRECTION	2600	UINT16	R/W	
EIO_DIRECTION	2601	UINT16	R/W	
CIO_DIRECTION	2602	UINT16	R/W	
MIO_DIRECTION	2603	UINT16	R/W	

FIO_DIRECTION
Read or write the direction of the 8 bits of FIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

EIO_DIRECTION
Read or write the direction of the 8 bits of EIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

CIO_DIRECTION
Read or write the direction of the 4 bits of CIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

MIO_DIRECTION
Read or write the direction of the 3 bits of MIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

For example: To set FIO1-7 to output, write a value of 0x01FF to FIO_DIRECTION. FIO0 is the least significant bit, so to prevent modification the corresponding inhibit bit is set with 0x01 in the most significant byte. The least significant byte is 0xFF, which is all 8 bits of FIO set to output.

Alternative DIO Registers

The DIO registers are the same thing as EIO, FIO, CIO, and MIO but with a more intuitive naming scheme, and a more compact register allotment.

DIO Registers

Name	Start Address	Type	Access	Default
DIO#(0:22)	2000	UINT16	R/W	
DIO_STATE	2800	UINT32	R/W	
DIO_DIRECTION	2850	UINT32	R/W	
DIO_INHIBIT	2900	UINT32	R/W	

DIO#(0:22)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
DIO0, DIO1, DIO2, Show All	2000, 2001, 2002, Show All

DIO_STATE
Read or write the state of all digital I/O in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes only apply to bits with mask set.

DIO_DIRECTION
Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes only apply to bits with mask set.

DIO_INHIBIT
A single binary-encoded value where each bit determines whether _STATE or _DIRECTION writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Electrical Overview

All digital I/O on the T7 have 3 possible states: input, output-high, or output-low. Each bit of I/O can be configured individually. When configured as an input, a bit has a ~100 kΩ pull-up resistor to 3.3 volts (all digital I/O are at least 5 volt tolerant). When configured as output-high, a bit is connected to the internal 3.3 volt supply (through a series resistor). When configured as output-low, a bit is connected to GND (through a series resistor).

See Appendix A for more details.

By default, the DIO lines are digital I/O, but they can also be configured as PWM Output, Quadrature Input, Counters, etc (see Extended Feature section of this Datasheet).

FIO vs. EIO vs. CIO vs. MIO

DIO is a generic name used for all digital I/O. The DIO are subdivided into different groups called FIO, EIO, CIO, and MIO.

Sometimes these are referred to as different "ports". For example, FIO is an 8-bit port of digital I/O and EIO is a different 8-bit port of digital I/O. The different names (FIO vs. EIO vs. CIO vs. MIO) have little meaning, and generally you can call these all DIO0-DIO22 and consider them all the same. There are a couple details unique to different ports:

- The source impedance of an FIO line is about 550 ohms, whereas the source impedance of EIO/CIO/MIO lines is about 180 ohms. Source impedance might be important when sourcing or sinking substantial currents, such as when controlling relays.
- The MIO lines are automatically controlled when using analog input channel numbers from 16 to 127. This is for controlling external multiplexers or the Mux80 expansion board.

Power-up Defaults

The default condition of the digital I/O can be configured by the user. From the factory, all digital I/O are configured as inputs by default. Note that even if the default for a line is changed to output-high or output-low, there could be a small time (milliseconds) during boot-up where all digital I/O are in the factory default condition.

Protection

All the digital I/O include an internal series resistor that provides overvoltage/short-circuit protection. These series resistors also limit the ability of these lines to sink or source current. Refer to the Digital I/O Specifications.

The fact that the digital I/O are specified as 5-volt tolerant means that 5 volts can be connected to a digital input without problems (see the actual limits in the specifications in Appendix A).

Increase logic level to 5V

In some cases, an open-collector style output can be used to get a 5V signal. To get a low set the line to output-low, and to get a high set the line to input. When the line is set to input, the voltage on the line is determined by a pull-up resistor. The T7 has an internal ~100k resistor to 3.3V, but an external resistor can be added to a different voltage. Whether this will work depends on how much current the load is going to draw and what the required logic thresholds are. Say for example a 10k resistor is added from EIO0 to VS. EIO0 has an internal 100k pull-up to 3.3 volts and a series output resistance of about 180 ohms. Assume the load draws just a few microamps or less and thus is negligible. When EIO0 is set to input, there will be 100k to 3.3 volts in parallel with 10k to 5 volts, and thus the line will sit at about 4.85 volts. When the line is set to output-low, there will be 180 ohms in series with the 10k, so the line will be pulled down to about 0.1 volts.

The surefire way to get 5 volts from a digital output is to add a simple logic buffer IC that is powered by 5 volts and recognizes 3.3 volts as a high input. Consider the CD74ACT541E from TI (or the inverting CD74ACT540E). All that is needed is a few wires to bring VS, GND, and the signal from the LabJack to the chip. This chip can level shift up to eight 0/3.3 volt signals to 0/5 volt signals and provides

high output drive current (+/-24 mA).

Note: DAC0, DAC1 channels on the T7 can be set to 5 volts, providing 2 output lines with such capability.

13.1 DIO Extended Features

Digital extended features measure and generate digital waveforms. Almost every digital I/O line can be assigned a feature and many can be active simultaneously. Features include things like PWM, Quadrature, and pulse generation. Features are assigned to DIOs using their type index, and configured using the options, and value registers.

The table below lists the features available on each DIO. The Digital I/O of the LabJack are on the top of the table, with the features to the left.

		FIO (0-7)								EIO (0-7)								CIO (0-3)				MIO (0-2)		
		DIO																						
Feature	**Index#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
PWM Out	**0**	✔		✔	✔	✔	✔																	
PWM Out with Phase	**1**	✔		✔	✔	✔	✔																	
Pulse Out	**2**	✔		✔	✔	✔	✔																	
Frequency In	**3,4**	✔	✔																					
Pulse Width In	**5**	✔	✔																					
Line-to-Line In	**6**	✔	✔																					
High-Speed Counter	**7**																	✔	✔	✔	✔			
Interrupt Counter	**8**	✔	✔	✔	✔			✔	✔															
Interrupt Counter with Debounce	**9**	✔	✔	✔	✔			✔	✔															
Quadrature In	**10**	✔	✔	✔	✔			✔	✔															
Interrupt Frequency In	**11**	✔	✔	✔	✔			✔	✔															
Digital I/O Extended Features																								

- **PWM Out** - Produces a rectangular output with variable frequency and variable duty cycle.
- **PWM Out with Phase** - Allows a phase difference between multiple PWM outputs.
- **Pulse Out** - You can specify the number of pulses, frequency of pulses, and pulse-width.
- **Frequency In** - Measures the period/frequency.
- **Pulse Width In** - Measures the high and low time, and thus also measures duty cycle.
- **Line-to-Line In** - Measures the time between edges on 2 different DIO lines.
- **High-Speed Counter** - Hardware-based edge counter.
- **Interrupt Counter** - A hardware edge counter that must service an interrupt for each edge.
- **Interrupt Counter with Debounce** - Use to avoid counting bounces from mechanical switches.
- **Quadrature In** - Tracks the forward/reverse count provided by a quadrature signal.
- **Interrupt Frequency In** - Frequency measurement that must service an interrupt for each edge.

All counters are 32-bit.

Each digital IO has a set of registers dedicated to the configuration of and results produced by the Extended Features. These registers are used to perform four operations on the Extended Feature: Configure, Read, Update, and Reset. Below you will find general descriptions of the four operations. Details about each feature can be found in their corresponding sections.

Configure:
Configuration is the initial setup of the Extended Feature. Configuration requires that any EF running at the pin in question first be disabled. Options can then be loaded. Then the EF can be enabled. The following seven registers are used for configuration:

DIO#_EF_ENABLE – 0 = Disable, 1 = Enable
DIO#_EF_INDEX – Index number specifying the Extended Feature
DIO#_EF_OPTIONS – Bits 2-0: Specifies the clock source to use
DIO#_EF_VALUE_A – Extended Feature specific value
DIO#_EF_VALUE_B – Extended Feature specific value
DIO#_EF_VALUE_C – Extended Feature specific value
DIO#_EF_VALUE_D – Extended Feature specific value

Read:
Some Extended Features produce results or provide status information that can be read. This information is usually a binary integer. When possible, the LabJack will convert the binary integer into a real-world unit such as seconds. When available, converted values can be read from the registers designated with "_F." The following registers are used to read results from a DIO Extended Feature:

DIO#_EF_READ_A – Extended feature specific value. Reading this value takes a snapshot of READ_B, READ_B_F.
DIO#_EF_READ_B – Extended feature specific value. Reading this returns the snapshot acquired by READ_A.
DIO#_EF_READ_A_F – Returns READ_A converted to a real-world value and takes a snapshot of READ_B, READ_B_F.
DIO#_EF_READ_B_F – Returns the READ_B snapshot converted to a real-world value.

Update:
Some Extended Features can be updated while running. Updating allows the Extended Feature to change its operation parameters without restarting. Note that the ClockSource and Feature Index can not be changed in an update. Depending on the feature, reads and writes to the update registers have small differences. See the Update portion of each features dedicated section for more information. The following four registers can be used to update an active Extended Feature:

DIO#_EF_VALUE_A – Extended feature specific value
DIO#_EF_VALUE_B – Extended feature specific value
DIO#_EF_VALUE_C – Extended feature specific value
DIO#_EF_VALUE_D – Extended feature specific value

Reset:
Some Extended Features can be reset while they are running. Resetting can have different results depending on the feature. For instance counters are reset to zero. There is only one register associated with resetting:

DIO#_EF_READ_A_AND_RESET – Extended feature specific value. Reading this resets the Extended Feature and takes a snapshot of READ_B so that it can be read as in the Read section. Values are read before the reset.
DIO#_EF_READ_A_F_AND_RESET – Returns the same information as DIO#_EF_READ_A_F. Reading this resets the Extended Feature and takes a snapshot of READ_B_F. Values are read before the reset.

Digital Extended Features

Name	Start Address	Type	Access	Default
DIO#(0:22)_EF_ENABLE	44000	UINT32	R/W	
DIO#(0:22)_EF_INDEX	44100	UINT32	R/W	
DIO#(0:22)_EF_OPTIONS	44200	UINT32	R/W	
DIO#(0:22)_EF_VALUE_A	44300	UINT32	R/W	
DIO#(0:22)_EF_VALUE_B	44400	UINT32	R/W	
DIO#(0:22)_EF_VALUE_C	44500	UINT32	R/W	
DIO#(0:22)_EF_VALUE_D	44600	UINT32	R/W	
DIO#(0:22)_EF_READ_A	3000	UINT32	R	
DIO#(0:22)_EF_READ_A_AND_RESET	3100	UINT32	R	
DIO#(0:22)_EF_READ_B	3200	UINT32	R	

DIO#(0:22)_EF_ENABLE
1 = enabled. 0 = disabled. Must be disabled during configuration.

Names	Addresses
DIO0_EF_ENABLE, DIO1_EF_ENABLE, DIO2_EF_ENABLE, Show All	44000, 44002, 44004, Show All

DIO#(0:22)_EF_INDEX
An index to specify the feature you want.

Names	Addresses
DIO0_EF_INDEX, DIO1_EF_INDEX, DIO2_EF_INDEX, Show All	44100, 44102, 44104, Show All

DIO#(0:22)_EF_OPTIONS
Function dependant on selected feature index.

Names	Addresses
DIO0_EF_OPTIONS, DIO1_EF_OPTIONS, DIO2_EF_OPTIONS, Show All	44200, 44202, 44204, Show All

DIO#(0:22)_EF_VALUE_A

Function dependant on selected feature index.

Names	Addresses
DIO0_EF_VALUE_A, DIO1_EF_VALUE_A, DIO2_EF_VALUE_A, Show All	44300, 44302, 44304, Show All

DIO#(0:22)_EF_VALUE_B
Function dependant on selected feature index.

Names	Addresses
DIO0_EF_VALUE_B, DIO1_EF_VALUE_B, DIO2_EF_VALUE_B, Show All	44400, 44402, 44404, Show All

DIO#(0:22)_EF_VALUE_C
Function dependant on selected feature index.

Names	Addresses
DIO0_EF_VALUE_C, DIO1_EF_VALUE_C, DIO2_EF_VALUE_C, Show All	44500, 44502, 44504, Show All

DIO#(0:22)_EF_VALUE_D
Function dependant on selected feature index.

Names	Addresses
DIO0_EF_VALUE_D, DIO1_EF_VALUE_D, DIO2_EF_VALUE_D, Show All	44600, 44602, 44604, Show All

DIO#(0:22)_EF_READ_A
Reads an unsigned integer value. The meaning of the integer is dependant on selected feature index.

Names	Addresses
DIO0_EF_READ_A, DIO1_EF_READ_A, DIO2_EF_READ_A, Show All	3000, 3002, 3004, Show All

DIO#(0:22)_EF_READ_A_AND_RESET
Reads the same value as DIO#(0:22)_EF_READ_A and forces a reset.

Names	Addresses
DIO0_EF_READ_A_AND_RESET, DIO1_EF_READ_A_AND_RESET, DIO2_EF_READ_A_AND_RESET, Show All	3100, 3102, 3104, Show All

DIO#(0:22)_EF_READ_B
Reads an unsigned integer value. The meaning of the integer is dependant on selected feature index.

Names	Addresses
DIO0_EF_READ_B, DIO1_EF_READ_B, DIO2_EF_READ_B, Show All	3200, 3202, 3204, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

How-To: Use a digital extended feature

1. Disable features on the DIO using ..._EF_ENABLE
2. Select a feature, and assign the corresponding type index to ..._EF_INDEX
3. Write to ..._EF_OPTIONS (if necessary)
4. Write to ..._EF_VALUE_A, ..._EF_VALUE_B, ..._EF_VALUE_C, ..._EF_VALUE_D (if necessary)
5. Enable feature on the DIO using ..._EF_ENABLE
6. Read results using ..._EF_READ_A, ..._EF_READ_B, or ..._EF_READ_A_AND_RESET

13.1.1 EF Clock Source

The ClockSources produce the reference frequencies used to generate output waveforms and measure input waveforms. ClockSource settings control output frequency, PWM resolution, maximum measurable period, and measurement resolution.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to

a divisor of 1.

There are 3 DIO EF clock sources available. Each clock source has an associated bit size and several mutual exclusions. Mutual exclusions exist because the clock sources share hardware with other features. A ClockSource is created form a hardware counter. CLOCK1 uses COUNTER_A (CIO0) and CLOCK2 uses COUNTER_B (CIO1). The 32-bit clock source (CLOCK0) is created by combining the 2 16-bit clock sources (CLOCK1 CLOCK2). The following list provides ClockSource bit sizes and mutual exclusions.

CLOCK0: 32-bit. Mutual Exclusions: CLOCK1, CLOCK2, COUNTER_A (CIO0), COUNTER_B(CIO1)
CLOCK1: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_A (CIO0)
CLOCK2: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_B (CIO1)

The clock source is not a DIO EF feature, but the four basic operations of Configure, Read, Update, and Reset still apply:

Configure:
There are four registers associated with the configuration of clock sources:
DIO_EF_CLOCK#_ENABLE: 1 = Enable, 0 = Disable. Must be disabled to change the configuration.
DIO_EF_CLOCK#_DIVISOR: 1, 2, 4, 8, 16, 32, 64, or 256. Default value of 0 equates to a divisor of 1.
DIO_EF_CLOCK#_OPTIONS: Reserved for future use. Write 0.
DIO_EF_CLOCK#_ROLL_VALUE: The ClockSource will count to VALUE-1 then roll to zero and repeat. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock. 0 results in the max roll value possible.

A ClockSource can be enabled after DIO EF_INDEX has been configured. This allows several DIO EFs to be started at the same time.

Read:
DIO_EF_CLOCK#_COUNT: Returns the current value of a clock source's counter. This can useful for generating timestamps.

Update:
A smooth update feature has been added in firmware 1.0035. Both the roll_value and the divisor can be written while a clock source is running. As long as the clock source's period is greater than 50 µs the clock will seamlessly switch to the new settings.

Reset:
At this time there are no reset operations available for the DIO EF clock sources.

Example:
Configure CLOCK0 as a 10 MHz clock source with a roll-value of 1000000.

DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 1000000
DIO_EF_CLOCK0_ENABLE = 1

With this clock configuration, PWM output (index=0) will have a frequency of 10 Hz. A frequency input measurement (index=3/4) will be able to count from 0-999999 with each count equal to 0.1 microseconds, and thus a max period of about 0.1 seconds.

Advanced:
If CLOCK0 is enabled and CLOCK1 and CLOCK2 are disabled, you can still select CLOCK1 or CLOCK2 as the source for a DIO EF channel. CLOCK1 CLOCK2 are actually the LSW & MSW of CLOCK0. The frequency of CLOCK1 is the same as CLOCK0. If DIO_EF_CLOCK0_ROLL_VALUE is >= 2^16, then the frequency of CLOCK2 is CLOCK0_freq divided by the modulus (remainder portion) of CLOCK0_freq / 2^16. If (CLOCK0_ROLL_VALUE - 1) is < 2^16, then the frequency of CLOCK2 is 0. CLOCK1_ROLL_VALUE is the modulus of (CLOCK0_ROLL_VALUE - 1) / 2^16 and CLOCK2_ROLL_VALUE is the quotient (integer portion) of (CLOCK0_ROLL_VALUE - 1) / 2^16.

Digital EF Clock Source

Name	Start Address	Type	Access	Default
DIO_EF_CLOCK0_ENABLE	44900	UINT16	R/W	
DIO_EF_CLOCK0_DIVISOR	44901	UINT16	R/W	
DIO_EF_CLOCK0_OPTIONS	44902	UINT32	R/W	
DIO_EF_CLOCK0_ROLL_VALUE	44904	UINT32	R/W	
DIO_EF_CLOCK0_COUNT	44908	UINT32	R	

DIO_EF_CLOCK0_ENABLE
1 = enabled. 0 = disabled. Must be disabled during configuration.

DIO_EF_CLOCK0_DIVISOR
Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

DIO_EF_CLOCK0_OPTIONS
Bitmask: bit0: 1 = use external clock. All other bits reserved.

DIO_EF_CLOCK0_ROLL_VALUE
The number of clock pulses to count before starting over at zero. The clock pulses counted are those after the divisor.

DIO_EF_CLOCK0_COUNT
Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

13.1.2 PWM Out

Capable DIO: **FIO0, FIO2, FIO3, FIO4, and FIO5**

Requires Clock Source: **Yes**

Index: **0**

PWM Out will generate a pulse width modulated wave form.

Operation:
PWM output will set the DIO high and low relative to the clock source's count. When the the count is zero the DIO line will be set high. When the count matches Config A the line will be set low. Therefore Config A is used to control the duty cycle and the resolution is equal to the roll value.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR // typically 80M/Divisor
PWMFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
*DutyCycle% = 100 * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE*

For the common case of CoreFrequency = 80 MHz we can rewrite output frequency as:

*PWMFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE*)

... and for 50% duty-cycle (square wave) simply set:

DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

PWM Out is capable of glitch-free updates in most situations. A glitch-free update means that the PWM will finish the current period consisting of the high time then the low time before loading the new value. The next period will then have the new duty cycle. This is true for all cases except zero. When setting the duty cycle to zero the line will be set low regardless of the current position. This means that a single high pulse with duration between zero and the previous high time can be output before the line goes low.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 0
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the specified clocks source's count matches this value the line will transition from high to low.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
The duty cycle can be updated at any time. To update, write the new value to DIO#_EF_CONFIG_A. The new value will not be used until the clock source rolls to zero. This means that at the end of the current period the new value will be loaded resulting in a glitch-free transition.

Read:
No information is returned by PWM Out

Reset:
Reset has no affect on this feature.

Example:
Generate a 10 kHz PWM starting at 25% DC.
First configure the clock source. The higher the roll value the greater the duty cycle resolution will be. For the highest resolution, we want to maximize the roll value, so use the smallest clock divisor that will not result in a roll value greater than the clock source's maximum (32-bits or 16-bits). With a divisor of 1 the roll value will be 8000: 80 MHz / (1 * 8000) = 10 kHz. Now set the registers accordingly:

DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 1
DIO_EF_CLOCK0_ROLL_VALUE = 8000
DIO_EF_CLOCK0_ENABLE = 1

Once the clock source is configured we can use the roll value to calculate Config_A: DC = 25% = 100 * Config_A / 8000. So Config_A = 2000. Now the PWM can be turned on by writing the proper registers:

DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 0
DIO0_EF_CONFIG_A = 2000
DIO0_EF_ENABLE = 1

Configure settings to generate LJM psuedocode

Desired FIO Channel:	FIO0
Desired Frequency (Hz):	1000
Desired Duty Cycle:	50
Select Clock Source:	Clock0
Select Clock Divisor:	1

Pseudocode:

```
// Configure Lua Script Functionality on T7
int handle = LJM_OpenS( "ANY", "ANY", "ANY");  // Open the first found device

// Start and Stop a Lua Script:
LJM_eWriteName(handle, "LUA_RUN", 1);  // Start a Lua Script
LJM_eWriteName(handle, "LUA_RUN", 0);  // Stop a Lua Script

// Saving Lua Script to a Device
// 1. Stop the Lua Script
LJM_eWriteName(handle, "LUA_RUN", 0); // Stop the Lua Script
string scriptData = "
LJ.IntervalConfig(0, 1000)
    while true do
    if LJ.CheckInterval(0) then
        print("Hello world, from ",_VERSION,"!\n")
    end
end";
int scriptLength = scriptData.length();
// 2. Write the size of the Lua Script to the device
LJM_eWriteName(handle, "LUA_SOURCE_SIZE", scriptLength);
// 3. Break up the Lua Script array into 32 byte chunks
//    and write them to LUA_SOURCE_WRITE
int numPackets = Math.ceil(scriptLength/32); // Calculate the number of packets to send
int numFullPackets = Math.floor(scriptLength/32); // Calculate the number of full packets to send
int numPartialPackets = numPackets - numFullPackets; // Calculate the number of partial packets to send
for(i = 0; i < numFullPackets; i++) {
    subStr = scriptData.split(i*32, (i+1)*32-1); // get the current packet's data
    dataArray = convertStrToArrayOfDoubles(subStr);
    LJM_eWriteNameArray(handle, "LUA_SOURCE_WRITE", 32, dataArray); // Write data to device
}
// 4. If necessary, write the remaining chunk of data
if(numPartialPackets > 0) {
    firstCharIndex = numFullPackets * 32;
```

```
    subStr = scriptData.split(firstCharIndex, scriptLength);
    dataArray = convertStrToArrayOfDoubles(subStr);
    int dataLength = dataArray.length();
    LJM_eWriteNameArray(handle, "LUA_SOURCE_WRITE", dataLength, dataArray); // Write data to device
}
```

Information regarding the functions used in this example are available on LJM's function reference page.

13.1.3 PWM Out with Phase

Capable DIO: **FIO0, FIO2, FIO3, FIO4, FIO5**

Requires Clock Source: **Yes**

Index: **1**

PWM Output with phase control generates PWM waveforms with the pulse positioned at different points in the period. This is achieved by setting the DIO line high and low relative to the clock source's count.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR // typically 80M/Divisor
PWMFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
*DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE*
*PhaseOffset = 360º * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE*

When the the count matches Value_B the DIO line will be set high. When the count matches Value A the line will be set low. Therefore Value_A minus Value_B controls the duty cycle.

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 1
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the clock source's count matches this value the line will transition from high to low.
DIO#_EF_CONFIG_B:When the clock source's count matches this value the line will transition from low to high.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

Update:
The duty cycle can be updated at any time. To update, write the new value to Config_A then Config_B. The value written to Config_A is stored until Config_B is written. After writing Config_B the new value will be loaded at the start of the next period. Updates are glitch-less unless switching from a very high to very low duty cycle or a very low to very high duty cycle.

DIO#_EF_CONFIG_A: Values written here will set the new falling position. The new value will not take effect until Config_B is written.
DIO#_EF_CONFIG_B: Values written here will set the new rising position. When Config_B is written the new Config_A is also loaded.

Read:
No information is returned by PWM Out with Phase.

Reset:
Reset has no affect on this feature.

13.1.4 Pulse Out

Capable DIO: **FIO0, FIO2, FIO3, FIO4, FIO5**

Requires Clock Source: **Yes**

Index: **2**

Pulse output will generate a specified number of pulses. The high time and the low time are specified relative to the clock source the same way as PWM with Phase Control.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR // typically 80M/Divisor
PulseOutFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
*DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE* // if A > B

For the common case of CoreFrequency = 80 MHz and Config-B fixed at 0, we can rewrite these as:

*PulseOutFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE*)
*DutyCycle% = 100 * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE*

... and thus for 50% duty cycle simply set:

DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Configure:
DIO#: First set the DIO line low (DIO#=0). The line must start low for proper pulse generation.
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 2
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the specified clock source's count matches this value the line will transition from high to low.
DIO#_EF_CONFIG_B: When the specified clock source's count matches this value the line will transition from low to high.
DIO#_EF_CONFIG_C: The number of pulses to generate.
DIO#_EF_CONFIG_D: Not used.

Update:
DIO#_EF_CONFIG_A: Sets a new high to low transition point. Will take effect when writing Config_C.
DIO#_EF_CONFIG_B: Sets a new low to high transition point. Will take effect when writing Config_C.
DIO#_EF_CONFIG_C: Writing to this value will start a new pulse sequence. If a sequence is already in progress it will be aborted. Numbers previously written to Config_A or Config_B will take effect when Config_C is written.

Read:
DIO#_EF_READ_A: The number of pulses that have been completed.
DIO#_EF_READ_B: The target number of pulses.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Reads number of pulses that have been completed. Then restarts the pulse sequence.

Example:
First configure a clock source to drive the pulse generator. Assuming the core frequency is 80 MHz writing the following registers will produce a 1 kHz pulse frequency.

DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1

Thus Clock0Frequency = 80 MHz / 8 = 10 MHz, and PWMFrequency = 10 MHz / 10000 = 1 kHz.

Now that we have a clock to work with we can configure our pulse.

```
DIO0_EF_ENABLE = 0
DIO0 = 0                    // set DIO0 to output-low
DIO0_EF_INDEX = 2           // pulse out type index
DIO0_EF_CONFIG_A = 2000     // high to low count
DIO0_EF_CONFIG_B = 0        // low to high count
DIO0_EF_CONFIG_C = 5000     // number of pulses
DIO0_EF_ENABLE = 1
```

Thus duty cycle = 100 * (2000 - 0) / 10000 = 20%. The LabJack will now output 5000 pulses over 5 seconds at 20% duty cycle.

13.1.5 Frequency In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **3 (positive edges) or 4 (negative edges)**

Frequency In will measure the period/frequency of a digital input signal by counting the number of clock source ticks between two edges ... rising-to-rising (index=3) or falling-to-falling (index=4). The number of ticks can be read from DIO#_EF_READ_A.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
Period (s) = DIO#_EF_READ_A / Clock#Frequency
Frequency (Hz) = Clock#Frequency / DIO#_EF_READ_A

Resolution (s) = 1 / Clock#Frequency
Max Period (s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be using a lower roll value for another feature such as PWM output.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

Divisor = 1, Resolution = 12.5 nanoseconds, MaxPeriod = 53.7 seconds
Divisor = 256, Resolution = 3.2 microseconds, MaxPeriod = 229 minutes

Once this feature is enabled, a new measurement happens on every applicable edge and the result registers are updated. If you do another read before a new edge has occurred, you will get the same value as before. Many applications will want to use the read-and-reset option so that a value is only read once and extra reads will return 0.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 3 or 4
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: Bit 1: 1=continuous, 0=OneShot. All other bits reserved.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed on Frequency In.

Read:
DIO#_EF_READ_A: Returns the period in ticks. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B: Starting with firmware 1.0114, READ_B will return the same value as READ_A. This is a capture register ... it is only updated when one of the READ_A registers is read.
DIO#_EF_READ_A_F: Returns the period in seconds. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B_F: Returns the frequency in Hz. If a full period has not yet been observed this value will be zero. This is a capture register ... it is only updated when one of the READ_A registers is read.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured.

Example:
First, configure the clock source. Roll value would usually be set to 0 to provide the maximum measurable period, but assume for this example that we have to use 10000 because of PWM output on another channel:

DIO_EF_CLOCK0_DIVISOR = 8 // Clock0Frequency = 80M/8 = 10 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1

This clock configuration results in Resolution = 1 / 10M = 0.1 us and MaxPeriod = 10000 / 10M = 1 ms.

Now configure the DIO_EF on FIO0 as frequency input.

DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 3 or 4 // Select rising or falling edges.
DIO#_EF_OPTIONS = 0 // Select the clock source.
DIO0_EF_ENABLE = 1 // Turn on the DIO_EF

At this point the LabJack is watching the IO lines for the specified edge. After the first two edges have been observed the time between them is stored, and this repeats for each subsequent edge. Results can be read from the READ registers defined above.

13.1.6 Pulse Width In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **5**

Pulse Width In will measure the high time and low time of a digital input signal, by counting the number of clock source ticks while the signal is high and low. The number of high ticks can be read from DIO#_EF_READ_A and the number of low ticks can be read from DIO#_EF_READ_B.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
HighTime (s) = DIO#_EF_READ_A / Clock#Frequency
LowTime (s) = DIO#_EF_READ_B / Clock#Frequency

Resolution (s) = 1 / Clock#Frequency
Max High or Low Time (s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be

using a lower roll value for another feature such as PWM output.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

Divisor = 1, Resolution = 12.5 nanoseconds, MaxTime = 53.7 seconds
Divisor = 256, Resolution = 3.2 microseconds, MaxTime = 229 minutes

Once this feature is enabled, a new measurement happens on every applicable edge and both result registers are updated on every rising edge. If you do another read before a new rising edge has occurred, you will get the same values as before. Many applications will want to use the read-and-reset option so that a value is only read once and extra reads will return 0.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 5
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: Bit 1: 1=continuous, 0=OneShot. All other bits reserved.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed on Pulse Width In.

Read:
DIO#_EF_READ_A: Returns the measured high time in clock source ticks and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B: Returns the measured low time in clock source ticks. This is a capture register ... it is only updated when one of the READ_A registers is read.
DIO#_EF_READ_A_F: Returns the measured high time in seconds and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B_F: Returns the measured low time in seconds. This is a capture register ... it is only updated when one of the READ_A registers is read.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Performs the same read as described above, but then also clears the register so that zero is returned until another full period is measured.

DIO#_EF_READ_A_F_AND_RESET: Performs the same read as described above, but then also clears the register so that zero is returned until another full period is measured.

Example:
First, configure the clock source. Roll value would usually be set to 0 to provide the maximum measurable period, but assume for this example that we have to use 10000 because of PWM output on another channel:

```
DIO_EF_CLOCK0_DIVISOR = 8    // Clock0Frequency = 80M/8 = 10 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in Resolution = 1 / 10M = 0.1 us and MaxPeriod = 10000 / 10M = 1 ms.

Now configure the DIO_EF on FIO0 as pulse width input.

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 5       // Pulse width input feature.
DIO0_EF_OPTIONS = 0  // Set to use clock source zero.
DIO0_EF_ENABLE = 1    // Enable the DIO_EF
```

After a full period (rising edge, falling edge, rising edge) has occurred the values are stored in the result registers, and this repeats at each rising edge. READ_A and READ_A_F both return the high time and save the low time that can be read from READ_B and READ_B_F. This ensures that both readings are from the same waveform cycle.

13.1.7 Line-to-Line In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **6**

Line-to-Line In measures the time between an edge on one DIO line and an edge on another DIO line by counting the number of clock source ticks between the two edges. The edges can be individually specified as rising or falling.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
Time (s) = DIO#_EF_READ_A / Clock#Frequency

Resolution (s) = 1 / Clock#Frequency
Max Time (s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be using a lower roll value for another feature such as PWM output.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

Divisor = 1, Resolution = 12.5 nanoseconds, MaxPeriod = 53.7 seconds

Divisor = 256, Resolution = 3.2 microseconds, MaxPeriod = 229 minutes

Line-to-Line In operates in a one-shot mode. Once the specified combination of edges is observed the data is saved and measuring stops. Another measurement can be started by resetting or performing the configuration procedure again.

Configure:
Configuring Line-to-Line In requires configuring two digital I/O lines (FIO0 & FIO1 only) as Line-to-Line feature index 6. The first DIO configured should be the one expecting the first edge. Any extended features on either DIO should be disabled before beginning configuration.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 6
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: 0 = falling edge. 1 = rising edge.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed on Line-to-Line In.

Read:
DIO#_EF_READ_A: Returns the one-shot measured time in clock source ticks. If the specified combination of edges has not yet been observed this value will be zero. DIO0 & DIO1 both return the same value.
DIO#_EF_READ_A_F: Returns the time in seconds.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Performs the same operation as DIO#_EF_READ_A, then clears the result and starts another measurement.

Example:
First, configure the clock source:

DIO_EF_CLOCK0_DIVISOR = 1 // Clock0Frequency = 80M/1 = 80 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 0
DIO_EF_CLOCK0_ENABLE = 1

This clock configuration results in Resolution = 1 / 80M = 12.5 ns and MaxPeriod = 2^32 / 80M = 53.7 seconds.

Now configure the DIO-EF on FIO0 & FIO1 as line-to-line:

DIO0_EF_ENABLE = 0
DIO1_EF_ENABLE = 0

DIO0_EF_INDEX = 6 // Index for line-to-line feature.
DIO0_EF_OPTIONS = 0 // Select the clock source.
DIO0_EF_CONFIG_A = 0 // Detect falling edge.
DIO0_EF_ENABLE = 1 // Turn on the DIO-EF

DIO1_EF_INDEX = 6 // Index for line-to-line feature.
DIO1_EF_OPTIONS = 0 // Select the clock source.
DIO1_EF_CONFIG_A = 0 // Detect falling edge.
DIO1_EF_ENABLE = 1 // Turn on the DIO-EF

At this point the device is watching FIO0 for a falling edge. Once that happens it watches for a falling edge on FIO1. Once that happens it stores the time between those 2 edges which you can read from the READ registers described above.

To do another measurement, repeat the DIO0-EF configuration above, or read from DIO0_EF_READ_A_AND_RESET.

13.1.8 High-Speed Counter

Capable DIO: **CIO0, CIO1, CIO2, CIO3**

Requires Clock Source: **No**

Index: **7**

The T7 supports up to 4 high-speed rising-edge counters that use hardware to achieve high count rates. These counters are shared with other resources as follows:

CounterA (DIO16/CIO0): Used by EF Clock0 & Clock1.
CounterB (DIO17/CIO1): Used by EF Clock0 & Clock2.
CounterC (DIO18/CIO2): Always available.
CounterD (DIO19/CIO3): Used by stream mode.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 7
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_B: Not used.

Update:
No update operations can be performed with High-Speed Counter.

Read:
DIO#_EF_READ_A: Returns the current count which is incremented on each rising edge.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30us. If missed edges at this point are not acceptable, then do not use reset but rather just note the "virtual reset" counter value in software and subtract it from other values.

Example:
Enable CounterC on DIO18/CIO2:

DIO18_EF_ENABLE = 0
DIO18_EF_INDEX = 7
DIO18_EF_ENABLE = 1

Results can be read from the READ registers defined above.

Frequency Measurement:
Counters are often used to measure frequency by taking change in count over change in time:

Frequency = (CurrentCount - PreviousCount) / (CurrentTimestamp - PreviousTimestamp)

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same modbus packet as the counter reads. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other digital extended features are available to measure frequency by timing individual pulses rather than counting over time.

13.1.9 Interrupt Counter

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **8**

Interrupt Counter counts pulses on the associated IO line. This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 8
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Not used.

DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

Update:
No update operations can be performed on Interrupt Counter.

Read:
DIO#_EF_READ_A: Returns the current Count

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30us. If missed edges at this point can not be tollerated then reset should not be used.

Example:
Enable a counter on FIO0:

DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 8
DIO0_EF_ENABLE = 1

Results can be read from the READ registers defined above.

Frequency Measurement:
Counters are often used to measure frequency by taking change in count over change in time:

Frequency = (CurrentCount - PreviousCount) / (CurrentTimestamp - PreviousTimestamp)

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same modbus packet as the counter reads. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other digital extended features are available to measure frequency by timing individual pulses rather than counting over time.

Edge Rate Limits:

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.1.10 Interrupt Counter with Debounce

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **9**

Requires firmware 1.0119 or higher.

Interrupt Counter with Debounce will increment its count by 1 when it receives a rising edge, a falling edge, or any edge (2x counting). After seeing an applicable edge, any further edges will be ignored during the debounce time.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Debounce Modes (DIO#_EF_CONFIG_B):
The exact behavior of the counting/debouncing is controlled by an index value written to DIO#_EF_CONFIG_B.

- 0: Count falling, debounce all, self-restarting timeout.
- 1: Count rising, debounce all, self-restarting timeout.
- 2: Count & debounce all, self-restarting timeout.
- 3: Count & debounce falling, fixed timeout.
- 4: Count & debounce rising, fixed timeout.
- 5: Timeout starts on falling edge. During timeout, a rising edge cancels and a falling edge restarts the timeout. After timeout any edge causes a count.
- 6: Timeout starts on rising edge. During timeout, a falling edge cancels and a rising edge restarts the timeout. After timeout any edge causes a count.

Self-restarting timeout means that during timeout any edge will restart the timeout with the value specified with DIO#_EF_CONFIG_A.

Mode 0 is commonly used with a normally-open push-button switch that is actuated by a person. We only want to count the push (falling edge), but expect bounce on the push & release (falling & rising edges) so need to debounce both.

Mode 4 might be used with some sort of device that outputs a fixed length positive pulse. For example, say a device provides a 1000 µs pulse, and there is always at least 5000 µs between pulses. Set the debounce timeout to 2000 µs so that the timeout period safely covers the entire pulse, but the timeout will for sure be done before another pulse can occur.

Modes 5 & 6 implement a requirement that the state of the line must remain low or high for some amount of time. For example, if you use mode 5 with a push-button switch and set DIO#_EF_CONFIG_A = 50000, that means that someone must push the switch and hold it down solidly for at least 50ms, and then the count will occur when they release the switch. An advantage to these modes is that they will ignore brief transient signals.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 9
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Debounce timeout in microseconds (µs).
DIO#_EF_CONFIG_B: Debounce mode index.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

Update:
No update operations can be performed on Interrupt Counter with Debounce.

Read:
DIO#_EF_READ_A: Returns the current Count

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer

READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30us. If missed edges at this point can not be tolerated then reset should not be used.

Example:
Enable a debounce counter on FIO0:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 9
DIO0_EF_CONFIG_A = 5000    // 5 ms debounce time
DIO0_EF_CONFIG_B = 0       // count falling, debounce all, self-restarting timeout
DIO0_EF_ENABLE = 1
```

Results can be read from the READ registers defined above.

Edge Rate Limits:

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.1.11 Quadrature In

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **10**

Quadrature input uses two DIOs to track a quadrature signal. Quadrature is a directional count often used in rotary encoders when you need to keep track of absolute position with forward & reverse movement. If you have movement in only one direction, or another way to know direction, you can simply use a normal counter with one phase of the encoder output.

The T7 uses 4x quadrature decoding, meaning that every edge observed (rising & falling on both phases) will increment or decrement the count.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Quadrature is prone to error if the edge rate is exceeded. This is particularly likely during direction change where the time between edges can be very small. Errors can occur when two edges come in too quickly for the device to process, which can result in missed counts or missed change in direction. These errors will be recorded and the quantity encountered can be read. If three edges come in

too quickly an undetectable error can occur.

Some quadrature encoders also include a third output channel, called a zero(z-phase) or index or reference signal, which supplies a single pulse per revolution. This single pulse is used for precise determination of a reference position. The T7 supports resets according to this reference signal. Z-phase will reset the the count when a high state is detected on the specified DIO line. If set to one-shot mode, Z-phase will clear the count only once. If the reference pulse is wider than A and B pulses, consider using the Conditional Reset feature instead of Z-phase.

Configure:
Configuring the T7 for quadrature requires configuring two DIO. The two lines must be an adjacent even/odd pair. FIO0/1, FIO2/3, and FIO6/7 are valid pairs. The even IO line will be phase A and the odd will be phase B.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 10
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Z-phase control: 0 = Z-phase off, 1 = Z-phase on, 3 = Z-phase on in one shot mode.
DIO#_EF_CONFIG_B: Z-phase DIO number.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed with Quadrature In.

Read:
DIO#_EF_READ_A - Returns the current count as a signed 2's complement value.
DIO#_EF_READ_B – Returns the number of detected errors.
DIO#_EF_READ_A_F - Starting with firmware 1.0114, READ_A_F will return the count in a single precision float (float32).

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET – Performs the same operation as DIO#_EF_READ_A, then sets the count to zero.

Example:
Configure FIO2 & FIO3 as quadrature inputs:

```
DIO2_EF_ENABLE = 0    //Make sure DIO-EF is disabled on FIO2
DIO3_EF_ENABLE = 0    //Make sure DIO-EF is disabled on FIO3

DIO2_EF_INDEX = 10    //Set feature index for FIO2 to quadrature.
DIO3_EF_INDEX = 10    //Set feature index for FIO3 to quadrature.

DIO2_EF_ENABLE = 1    //Enable quadrature DIO-EF on FIO2, for A phase.
DIO3_EF_ENABLE = 1    //Enable quadrature DIO-EF on FIO3, for B phase.
```

Edges on the two lines will now be decoded and the count will be incremented or decremented according to the edge sequence.

The current count can be read from:
DIO2_EF_READ_A or DIO2_EF_READ_A_AND_RESET

Testing:
On some LabJack devices you can randomly tap a GND wire into FIO2 and FIO3 to cause some counts, but that does not work on the T7. Testing needs to be done with a proper quadrature signal.

If testing with an actual encoder, first start with DIO-EF enabled and simply watch (e.g. in Kipling) the digital inputs as you slowly turn the encoder to see if the inputs change between high and low. That confirms a valid electrical connection.

You can test using 2 digital inputs to create a quadrature sequence. For example, configure quadrature on FIO2/3 as shown above, and connect FIO0 to FIO2 and FIO1 to FIO3.

FIO0 = 1 //Initialize FIO0 to output-high.
FIO1 = 1 //Initialize FIO1 to output-high.

DIO2_EF_ENABLE = 0 //Make sure DIO-EF is disabled on FIO2
DIO3_EF_ENABLE = 0 //Make sure DIO-EF is disabled on FIO3
DIO2_EF_INDEX = 10 //Set feature index for FIO2 to quadrature.
DIO3_EF_INDEX = 10 //Set feature index for FIO3 to quadrature.
DIO2_EF_ENABLE = 1 //Enable quadrature DIO-EF on FIO2, for A phase.
DIO3_EF_ENABLE = 1 //Enable quadrature DIO-EF on FIO3, for B phase.

Now we can simulate a quadrature signal by toggling FIO0 and 1 in the proper sequence. We will use the FIO_STATE register (address=2500) which operates on all 8 FIO bits at once, but we will set the inhibit bits for FIO2-FIO7 (bits 10-15) so they are not affected. To set bits 10-15 we can simply add 64512 to the desired FIO0/1 state value:

Write FIO_STATE = 3 + 64512, then should read DIO2_EF_READ_A = 0.
Write FIO_STATE = 1 + 64512, then should read DIO2_EF_READ_A = 1.
Write FIO_STATE = 0 + 64512, then should read DIO2_EF_READ_A = 0.
Write FIO_STATE = 2 + 64512, then should read DIO2_EF_READ_A = -1.
Write FIO_STATE = 3 + 64512, then should read DIO2_EF_READ_A = -2.
Write FIO_STATE = 2 + 64512, then should read DIO2_EF_READ_A = -1.
Write FIO_STATE = 0 + 64512, then should read DIO2_EF_READ_A = 0.
Write FIO_STATE = 1 + 64512, then should read DIO2_EF_READ_A = 1.
Write FIO_STATE = 3 + 64512, then should read DIO2_EF_READ_A = 2.
Write FIO_STATE = 2 + 64512, then should read DIO2_EF_READ_A = 3.
Write FIO_STATE = 0 + 64512, then should read DIO2_EF_READ_A = 4.
Write FIO_STATE = 1 + 64512, then should read DIO2_EF_READ_A = 5.
Write FIO_STATE = 3 + 64512, then should read DIO2_EF_READ_A = 6.
Write FIO_STATE = 2 + 64512, then should read DIO2_EF_READ_A = 7.

Edge Rate Limits:

Keep in mind that the T7 does 4x quadrature counting, so for example a 100 pulses/revolution encoder will generate 400 edges/revolution.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.1.12 Interrupt Frequency In

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, FIO7**

Requires Clock Source: **No ... uses core clock / 2.**

Index: **11**

Interrupt Frequency In will measure the frequency of a signal on the associated DIO line.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

To measure the frequency the LabJack will measure the duration of one or more periods. There are several option available to control the way the LabJack does this. The number of period to be averaged, the edge direction to trigger on and whether to measure continuously or in a one-shot mode can all be specified.

The clock source for this feature is simply half the core frequency:

ClockFrequency = CoreFrequency / 2 //Typically 80M/2 = 40 MHz

Period (s) = DIO#_EF_READ_A / ClockFrequency
Frequency (Hz) = ClockFrequency / DIO#_EF_READ_A

The maximum measurable time is 107 s. The number of periods to be averaged times the maximum expected period must be less than 107 s or the result will overflow: 107 < (NumToAverage * MaxPeriod)

By default Interrupt Frequency In will measure the frequency once and return that same result until it is reconfigured or reset. At which point a second measurement will be made. The other option is continuous mode. In continuous mode the frequency is constantly being measured and read returns the most recent result. Running continuous puts a greater load on the processor.

Configure:
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 11
DIO#_EF_OPTIONS: Not Used.
DIO#_EF_CONFIG_A: bit 0: Edge select; 1 = rising, 0 = falling. Bit 1: 1=continuous, 0=OneShot.
DIO#_EF_CONFIG_B: Number of periods to be measured.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed with Interrupt Frequency In.

Read:
DIO#_EF_READ_A: Returns the average period per cycle in ticks (core clock ticks / 2).
DIO#_EF_READ_B: Starting with firmware 1.0114, READ_B returns the total core clock tick count.
DIO#_EF_READ_A_F: Returns the average period per cycle in seconds. Takes into account the number of periods to be averaged and the core clock speed.
DIO#_EF_READ_B_F: Returns the average frequency in Hz. Takes into account the number of periods to be averaged and the core clock speed.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset:
DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured.
DIO#_EF_READ_A_AND_RESET_F: Returns the same data as DIO#_EF_READ_A_F and then clears the result so that zero is returned by subsequent reads until another full period is measured.

Example:

Edge Rate Limits:

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate

limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.1.13 Conditional Reset

DIO_EF Conditional Reset will reset a specified DIO_EF after a specified number of edges have been detected.

Configure:.
To set up a DIO_EF Conditional Reset is simple. Just set the DIO num of the DIO_EF you would like to reset and then set the other options. More options are likely to be added, so be sure to leave unused bits cleared to zero.
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 12
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Reset Options bitmask:

- bit 0: Edge select. 1 = rising, 0 = falling
- bit 1: reserved
- bit 2: OneShot. 1 = only reset once. 0 = reset every n edges.

DIO#_EF_CONFIG_B: Number of edges per reset.
DIO#_EF_CONFIG_C: IO number of DIO_EF to be reset.
DIO#_EF_CONFIG_D: Not used.

Update:
No update operations can be performed on Conditional Reset.

Read:
DIO#_EF_READ_A – Returns the current count.
DIO#_EF_READ_B – Not Used.
DIO#_EF_READ_A_F – Not Used.
DIO#_EF_READ_A_AND_RESET – Not Used.

Stream Read:
All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Example:
This example assumes that DIO0 has a running extended feature such as quadrature or a counter. Now we will set up DIO2 as a falling edge trigger that will reset the count of DIO0_EF. DIO2_EF_ENABLE = 0 // Ensure that the DIO_EF is not running so that it can be configured. DIO2_EF_INDEX = 12 // Set to Conditional Reset DIO2_EF_CONFIG_A = 0 // Falling edges DIO2_EF_CONFIG_B = 1 // Reset every edges DIO2_EF_CONFIG_C = 0 // Reset events clear the count of DIO0_EF DIO2_EF_ENABLE = 1 // Turn on the DIO_EF Now falling edges on DIO2 will set the count of DIO0_EF to zero.

13.2 I2C

The T7 supports Inter-Integrated Circuit (I²C or I2C) communication as the master only. I²C is a synchronous serial protocol typically used to communicate with chips that support I2C as slave devices. Any 2 digital I/O lines are used for SDA and SCL. Note that the I²C bus generally requires pull-up resistors of perhaps 4.7 kΩ from SDA to Vs and SCL to Vs, and also note that the screw terminals labeled SDA and SCL (if present) are not used for I²C.

This serial link is not an alternative to the USB connection. Rather, the host application will write/read data to/from the T7, and the T7 communicates with some other device using the serial protocol. Using this serial protocol is considered an advanced topic. A good knowledge of the protocol is recommended, and a logic analyzer or oscilloscope might be needed for troubleshooting.

I2C Registers

Name	Start Address	Type	Access	Default
I2C_SDA_DIONUM	5100	UINT16	R/W	0
I2C_SCL_DIONUM	5101	UINT16	R/W	0
I2C_SPEED_THROTTLE	5102	UINT16	R/W	0
I2C_SLAVE_ADDRESS	5104	UINT16	R/W	0
I2C_NUM_BYTES_TX	5108	UINT16	R/W	0
I2C_NUM_BYTES_RX	5109	UINT16	R/W	0
I2C_OPTIONS	5103	UINT16	R/W	0
I2C_GO	5110	UINT16	R/W	0
I2C_ACKS	5114	UINT32	R/W	0
I2C_DATA_TX	5120	BYTE	R/W	0
I2C_DATA_RX	5160	BYTE	R/W	0

I2C_SDA_DIONUM
The number of the DIO line to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

I2C_SCL_DIONUM
The number of the DIO line to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

I2C_SPEED_THROTTLE
This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz, 65516 is ~100 kHz.

I2C_SLAVE_ADDRESS
The 7-bit address of the slave device. Value is shifted left by firmware to allow room for the I2C R/W bit.

I2C_NUM_BYTES_TX
The number of data bytes to transmit.

I2C_NUM_BYTES_RX
The number of data bytes to read.

I2C_OPTIONS
Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and a start, 1 = Resetarts will not use a stop. bit 2: 1 = disable clock stretching.

I2C_GO
Writing to this register will instruct the LabJack to perform an I2C transaction.

I2C_ACKS
An array of bits used to observe ACKs from the slave device.

I2C_DATA_TX
Data that will be written to the I2C bus.

I2C_DATA_RX
Data that has been read from the I2C bus.

13.2.1 Configuration Registers

I2C Registers

Name	Start Address	Type	Access	Default
I2C_SDA_DIONUM	5100	UINT16	R/W	0
I2C_SCL_DIONUM	5101	UINT16	R/W	0
I2C_SPEED_THROTTLE	5102	UINT16	R/W	0
I2C_OPTIONS	5103	UINT16	R/W	0
I2C_SLAVE_ADDRESS	5104	UINT16	R/W	0

I2C_NUM_BYTES_TX	5108	UINT16	R/W	0
I2C_NUM_BYTES_RX	5109	UINT16	R/W	0
I2C_WRITE_DATA	5120	BYTE	R/W	0
I2C_READ_DATA	5160	BYTE	R/W	0
I2C_GO	5110	UINT16	R/W	0
I2C_ACKS	5114	UINT32	R/W	0

I2C_SDA_DIONUM
The number of the DIO line to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

I2C_SCL_DIONUM
The number of the DIO line to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

I2C_SPEED_THROTTLE
This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz, 65516 is ~100 kHz.

I2C_OPTIONS
Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and a start, 1 = Resetarts will not use a stop. bit 2: 1 = disable clock stretching.

I2C_SLAVE_ADDRESS
The 7-bit address of the slave device. Value is shifted left by firmware to allow room for the I2C R/W bit.

I2C_NUM_BYTES_TX
The number of data bytes to transmit.

I2C_NUM_BYTES_RX
The number of data bytes to read.

I2C_WRITE_DATA
Data that will be written to the I2C bus.

I2C_READ_DATA
Data that has been read from the I2C bus.

I2C_GO
Writing to this register will instruct the LabJack to perform an I2C transaction.

I2C_ACKS
An array of bits used to observe ACKs from the slave device.

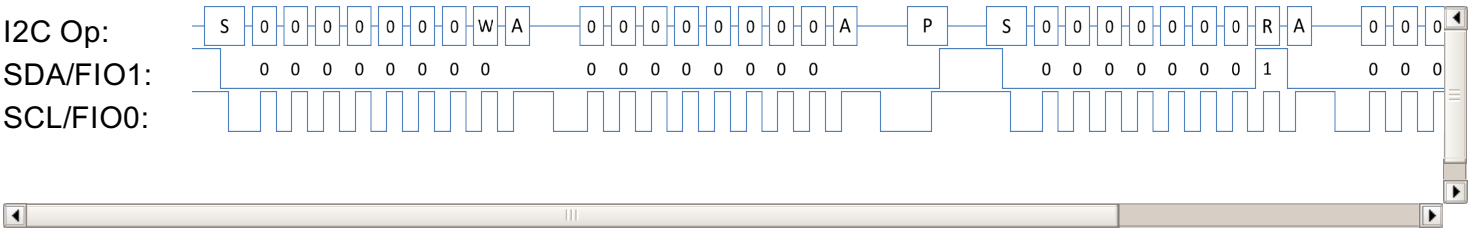
13.2.1 I2C Simulation Tool

Click HERE for a larger version of this tool

This javascript application is designed to help give an understanding of the LabJack's I2C functionality. Each I2C register that effects the output is shown below. The two registers that are omitted are "I2C_GO" and "I2C_ACKS".

"I2C_GO" Executes the configured I2C request, and "I2C_ACKS" reads the received ack's & nack's packed into a binary array.

Test out various I2C configuration settings and view the expected result.



Enter I2C Configuration Settings Below

I2C_SDA_DIONUM:	1
I2C_SCL_DIONUM:	0
I2C_SPEED_THROTTLE:	0

I2C_OPTIONS:	Enable Reset-On-Start:	☐
	Enable No-Stop on Restart:	☐
	Enable Clock Stretching:	☐
	Result:	0

--Input below in hexadecimal--

I2C_SLAVE_ADDRESS:	0
I2C_NUM_BYTES_TX:	1
I2C_NUM_BYTES_RX:	1
I2C_WRITE_DATA:	0
I2C_READ_DATA:	xx

Table Of Contents:

- **A:** Indicates an I2C-Ack bit writing a byte of data
- **C:** Indicates an I2C-Clock stretch occuring to slow down transmission
- **N:** Indicates an I2C-Nack bit writing a byte of data
- **P:** Indicates an I2C-Stop Condition
- **R:** Indicates an I2C-Read bit appearing after the slave address is sent
- **RESET:** Indicates an I2C-Reset Condition
- **S:** Indicates an I2C-Start Condition
- **W:** Indicates an I2C-Write bit appearing after the slave address is sent

Have Synchronous I2C Stop Condition below

13.3 SPI

The T7 supports Serial Peripheral Interface (SPI) communication as the master only. SPI is a synchronous serial protocol typically used to communicate with chips that support SPI as slave devices.

This serial link is not an alternative to the USB connection. Rather, the host application will write/read data to/from the T7, and the T7 communicates with some other device using the serial protocol. Using this serial protocol is considered an advanced topic. A good knowledge of the protocol is recommended, and a logic analyzer or oscilloscope might be needed for troubleshooting.

Example:
This is a simple loop-back test. Connect a jumper between DIO2 and DIO3. Data will be sent out DIO3 and read from DIO2.

SPI_CS_DIONUM = 0 // Use DIO0 as chip select.
SPI_CLK_DIONUM = 1 // Use DIO1 as Clock
SPI_MISO_DIONUM = 2 // Use DIO2 as MISO
SPI_MOSI_DIONUM = 3 // Use DIO3 as MOSI
SPI_MODE = 0 // Select mode zero.
SPI_SPEED_THROTTLE = 65500 // Set clock speed to ~100 kHz.
SPI_OPTIONS = 0 // No special operation
SPI_NUM_BYTES = 1 // Transfer one byte.
SPI_DATA_TX = 0x55 // Load a test data byte. eWriteNameArray is the easiest way to write the outgoing SPI data.
SPI_GO = 1 // Initiate the transfer

At this point the T7 will run the SPI transaction. If no errors are returned the received data can be read.

SPI_DATA_RX = ? - Read a test data byte. eReadNameArray is the easiest way to read the received SPI data.

Because of the loop-back the data read from SPI_DATA_RX will match the data written to SPI_DATA_TX.

Following are approximate clock rates measured for various values of SPI_SPEED_THROTTLE with firmware 1.0150:

0 780 kHz
65530 380 kHz
65500 100 kHz
65100 10 kHz
61100 1 kHz
21000 100 Hz

1 67 Hz

The T7 has in internal watchdog that will timeout, and cause the T7 to reboot, if a single SPI transaction lasts longer than 250ms. Stay safely above the following measured minimum values where device did not reboot for different numbers of bytes:

#Bytes	ThrottleValue	ClockRateHz
1	1	67 Hz (minimum)
2	4900	73 Hz
3	23900	106 Hz
4	33900	140 Hz
10	52600	342 Hz
16	57400	544 Hz
32	61500	1095 Hz

SPI Registers

Name	Start Address	Type	Access	Default
SPI_CS_DIONUM	5000	UINT16	R/W	0
SPI_CLK_DIONUM	5001	UINT16	R/W	0
SPI_MISO_DIONUM	5002	UINT16	R/W	0
SPI_MOSI_DIONUM	5003	UINT16	R/W	0
SPI_MODE	5004	UINT16	R/W	0
SPI_SPEED_THROTTLE	5005	UINT16	R/W	0
SPI_OPTIONS	5006	UINT16	R/W	0
SPI_GO	5007	UINT16	W	0
SPI_NUM_BYTES	5009	UINT16	R/W	0
SPI_DATA_TX	5010	BYTE	W	0
SPI_DATA_RX	5050	BYTE	R	0

SPI_CS_DIONUM
The DIO line for Chip-Select.

SPI_CLK_DIONUM
The DIO line for Clock.

SPI_MISO_DIONUM
The DIO line for Master-In-Slave-Out.

SPI_MOSI_DIONUM
The DIO line for Master-Out-Slave-In.

SPI_MODE
The SPI mode controls the clock idle state and which edge clocks the data. Bit 1 is CPOL and Bit 0 is CPHA, so CPOL/CPHA for different decimal values: 0 = 0/0 = b00, 1 = 0/1 = b01, 2 = 1/0 = b10, 3 = 1/1 = b11.

SPI_SPEED_THROTTLE
This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~1 MHz. 1 results in ~100 Hz.

SPI_OPTIONS
Bit 0 is Auto-CS-Disable.

SPI_GO
Write 1 to begin the configured SPI transaction.

SPI_NUM_BYTES
The number of bytes to transfer.

SPI_DATA_TX
Write data here.

SPI_DATA_RX
Read data here.

13.4 SBUS

SBUS is a serial protocol used with SHT1X and SHT7x sensors from Sensirion. It is similar to I2C, but not the same. The EI-1050 uses the SHT11 sensor. Other available sensors are the SHT10, SHT15, SHT71, and SHT75.

SBUS Registers

Name	Start Address	Type	Access	Default
SBUS#(0:22)_TEMP	30100	FLOAT32	R	
SBUS#(0:22)_RH	30150	FLOAT32	R	
SBUS#(0:22)_DATA_DIONUM	30200	UINT16	R/W	0
SBUS#(0:22)_CLOCK_DIONUM	30225	UINT16	R/W	1
SBUS_ALL_DATA_DIONUM	30275	UINT16	R/W	0
SBUS_ALL_CLOCK_DIONUM	30276	UINT16	R/W	1
SBUS_ALL_POWER_DIONUM	30277	UINT16	R/W	2
SBUS_ALL_CLOCK_SPEED	30278	UINT16	R/W	65000

SBUS#(0:22)_TEMP
Reads temperature in degrees Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Names	Addresses
SBUS0_TEMP, SBUS1_TEMP, SBUS2_TEMP, Show All	30100, 30102, 30104, Show All

SBUS#(0:22)_RH
Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Names	Addresses
SBUS0_RH, SBUS1_RH, SBUS2_RH, Show All	30150, 30152, 30154, Show All

SBUS#(0:22)_DATA_DIONUM
This is the DIO# that the external sensor's data line is connected to. Default = FIO0

Names	Addresses
SBUS0_DATA_DIONUM, SBUS1_DATA_DIONUM, SBUS2_DATA_DIONUM, Show All	30200, 30201, 30202, Show All

SBUS#(0:22)_CLOCK_DIONUM
This is the DIO# that the external sensor's clock line is connected to. Default = FIO1

Names	Addresses
SBUS0_CLOCK_DIONUM, SBUS1_CLOCK_DIONUM, SBUS2_CLOCK_DIONUM, Show All	30225, 30226, 30227, Show All

SBUS_ALL_DATA_DIONUM
A write to this global parameter sets all SBUS data line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

SBUS_ALL_CLOCK_DIONUM
A write to this global parameter sets all SBUS clock line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

SBUS_ALL_POWER_DIONUM
Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default = FIO2. An FIO line can power up to 4 sensors while an EIO/CIO/MIO line or DAC line can power up to 20 sensors. Set to 9999 to disable. To use multiple power lines, use a DAC line for power, or otherwise control power yourself, set this to 9999 and then control power using writes to normal registers such as FIO5, EIO0, or DAC0.

SBUS_ALL_CLOCK_SPEED
Sets the clock speed. Clock is software generated so the frequency is not fixed. Larger values are faster. Zero is the fastest option and is equivalent to 65536. A value of zero is ~200 kHz. a value of 65000 is ~9.1 kHz.

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Examples:

EI-1050 probes using default configuration:

The EI-1050 has an enable line that allows multiple probes to use the same pair of data/clock lines. In this example we connect the 4 basic wires from each probe to the lines specified by the default config:

GND Ground (black)
FIO0 Data (green)
FIO1 Clock (white)
FIO2 Power (red)

Since power is provided by FIO2, and FIO lines can only power 4 EI-1050 probes, that is the limitation on number of probes using the default config. We can now connect the enable line from each probe to any DIO we want. Lets use:

FIO3 Enable ProbeA (brown)
EIO0/DIO8 Enable ProbeB (brown)
EIO1/DIO9 Enable ProbeC (brown)
EIO2/DIO10 Enable ProbeD (brown)

You can now read from SBUS#_TEMP and SBUS#_RH for each probe without writing any config values. In LJLogM, for example, just put the desired register name in any row. A read from SBUS3_TEMP will return the temperature from ProbeA. A read from SBUS9_RH will return the humidity from ProbeC.

Note that when using multiple probes this way, you might need to read one value from each probe before they will work. By default digital I/O are set to input, which has a 100k pull-up, so all 4 probes in this example will be enabled at the same time, which will likely result in a read error. At the end of a read the enable line is set to output-low, so once you do an initial read from each, they will all be disabled and on further reads only one will be enabled at a time.

EI-1050 probes using enable with custom configuration:

Say you connect 2 probes as follows:

GND Ground (black)
EIO0/DIO8 Data (green)
EIO1/DIO9 Clock (white)
EIO2/DIO10 Power (red)

EIO3/DIO11 Enable ProbeA (brown)
EIO4/DIO12 Enable ProbeB (brown)

Write the following registers to configure and disable the probes:

SBUS_ALL_DATA_DIONUM = 8
SBUS_ALL_CLOCK_DIONUM = 9
SBUS_ALL_POWER_DIONUM = 10
EIO3 = 0
EIO4 = 0

You can now read from SBUS11_TEMP/SBUS11_RH for ProbeA values or SBUS12_TEMP/SBUS12_RH for ProbeB values.

EI-1050, SHT1x, or SHT7x, using individual data/clock lines and DAC0 for power:

Say you connect an EI-1050 and SHT71 as follows:

GND Ground (black)
DAC0 Power (red)

FIO0 Data for EI-1050 (green)
FIO1 Clock for EI-1050 (white)
DAC0 Enable EI-1050 (brown)
FIO2 Data for SHT71
FIO3 Clock for SHT71

Since the EI-1050 is tied to power, it will always be enabled. We can do that because we have assigned it dedicated DIO for data and clock. The SHT71 does not have an enable. Note that the SHT7x datasheet shows an added 10k pull-up resistor from Data to Power. The LabJack has an internal 100k pull-up that usually works, but some applications might need the stronger 10k pull-up and perhaps even a capacitor from Clock to GND near the sensor pins.

Write the following registers to configure and power the probes:

SBUS0_DATA_DIONUM = 0
SBUS0_CLOCK_DIONUM = 1
SBUS3_DATA_DIONUM = 2

SBUS3_CLOCK_DIONUM = 3
SBUS_ALL_POWER_DIONUM = 9999
DAC0 = 3.3

You can now read from SBUS0_TEMP/SBUS0_RH for the EI-1050 values or SBUS3_TEMP/SBUS3_RH for the SHT71 values.

Note that the "#" in the register names above can be about anything you want. Say for the SHT71 you instead did:

SBUS7_DATA_DIONUM = 2
SBUS7_CLOCK_DIONUM = 3

Now if you read SBUS7_TEMP/SBUS7_RH, the LabJack will use FIO2/3 to talk to the sensor. A possible problem, though, is that the LabJack will also control FIO7 as an enable. It will set FIO7 to output-high, talk to the sensor, and then set FIO7 to output-low. The way to prevent control of an enable line is to use a "#" that is the same as the data or clock line.

13.5 1-Wire

This document assumes that the reader has a basic understanding of the 1-wire protocol.

1-Wire is a serial protocol that uses only one data line. Multiple devices can be connected to a single 1-Wire bus and are differentiated using a unique 64-bit number referred to as ROM.

Hardware:
Devices on the 1-wire bus need to be connected to GND, Vs and the data line DQ. DQ also needs a pullup resister of 2.2-4.7 kΩ to Vs.

FIO lines can not be used for 1-Wire. They have too much impedance which prevent the signal from reaching logic thresholds.

The T7 supports a DPU (dynamic pull up). A dynamic pull up uses an external circuit such as a transistor to provide extra power to the DQ line at proper times. This can be helpful if the line is large or you are using parasitic power.

Configuration:
ONEWIRE_DQ_DIONUM: This is the DIO line to use for the data line, DQ.
ONEWIRE_DPU_DIONUM: This is the DIO line to use for the dynamic pullup control.
ONEWIRE_OPTIONS: A bit-mask for controlling operation details.
bit 0: Reserved, write 0.
bit 1: Reserved, write 0.
bit 2: DPU Enable. Write 1 to enable the dynamic pullup.
bit 3: DPU Polarity. Write 1 to set the active state as high, 0 to set the active state as low.
ONEWIRE_FUNCTION: This controls how the ROM address of 1-wire devices will be used.
ONEWIRE_NUM_BYTES_TX: The number of bytes to transmit to the device. Has no affect when the ROM function is set to Search or Read.
ONEWIRE_NUM_BYTES_RX:The number of bytes to read from the device. Has no affect when the ROM function is set to Search or Read.
ONEWIRE_ROM_MATCH_H: The upper 32-bits of the ROM of the device to attempt to connect to when using the Match ROM function.
ONEWIRE_ROM_MATCH_L: The lower 32-bits of the ROM of the device to attempt to connect to when using the Match ROM function.
ONEWIRE_PATH_H: Upper 32-bits of the search path.
ONEWIRE_PATH_L: Lower 32-bits of the search path.

ROM Functions:
0xF0: Search – This function will read the ROM of one device on the bus. The ROM found is placed in ONEWIRE_SEARCH_RESULT and if other devices were detected the branch bits will be set in ONEWIRE_ROM_BRANCHS_FOUND.
0xCC: Skip – This function will skip the ROM addressing step. For this to work properly only one device may be connected to the bus.
0x55: Match – When using this function data will be sent to and read from a device whose ROM matches the ROM loaded into the ONEWIRE_ROM_MATCH registers.
0x33: Read – Reads the ROM of the connected device. For this to work properly only one device may be connected to the bus.

Sending data:
When using the Match or Skip Rom functions data can be sent to the device. To do so, set the number of bytes to send by writing to ONEWIRE_NUM_BYTES_RX and write the data to ONEWIRE_DATA_RX.

Reading data:

When using the Match or Skip Rom functions data can be read from the device. To do so, set the number of bytes to send by writing to ONEWIRE_NUM_BYTES_TX and write the data to ONEWIRE_DATA_TX.

Example:
Configure the T7's 1-Wire interface, and obtain a temperature reading from a DS18B22.

Configuration:
Write the common configuration that will not change; the DQ line, DPU, and options. For this example we will use EIO6 (14) as DQ, and the DPU will be left disabled.

ONEWIRE_DQ_DIONUM = 14
ONEWIRE_DPU_DIONUM = 0
ONEWIRE_OPTIONS = 0

Read ROM:
The 64-bit ROM can be read from the device using the Read ROM function if it is the only device on the bus.

ONEWIRE_FUNCTION = 0x33
ONEWIRE_GO = 1

The T7 will read the ROM from the connected device and place it in ONEWIRE_SEARCH_RESULT. This test resulted in ROM code 0x1D000005908D4728

Search for ROM:
If there is more than one device on the bus the search function can be used to find the ROM of one of the devices. Note that this method does not provide any information about which device has the ROM discovered.

ONEWIRE_PATH = 0
ONEWIRE_FUNCTION = 0xF0
ONEWIRE_GO = 1

The LabJack will perform the 1-Wire search function. If a ROM is found it will be placed in ONEWIRE_ROM_SEARCH_RESULT and any branches detected will be indicated in ONEWIRE_BRANCES. The ONEWIRE_PATH field can be used to direct the LabJack to take a different path in subsequent searches.

Results:
ROM - 0x1D000005908D4728
Branches - 0x00000000000002
Now repeat the search with path set to 2.

Results:
ROM - 0xFF00000024AD2C22
Branches - 0x00000000000002

The search can be repeated to find the ROM codes of all devices on the bus.

Write start conversion command to the device:
Do instruct the sensor to start a reading we need to match the device's ROM and send one data byte. The data byte contains the instruction 0xBE.

ONEWIRE_FUNCTION = 0x55
ONEWIRE_ROM = 0x1D000005908D4728
ONEWIRE_NUM_BYTES_TX = 1
ONEWIRE_DATA_TX = [0x44]
ONEWIRE_GO = 1

The sensor will now start a conversion. Depending on the sensor and it's settings up to 500 ms may be needed to complete the conversion.

Read conversion result from the device:
After a conversion has been complete we can begin the reading process. This time we need to write the read instruction which is 0x44 and then read 9 bytes of data.

ONEWIRE_FUNCTION = 0x55
ONEWIRE_ROM = 0x1D000005908D4728
ONEWIRE_NUM_BYTES_TX = 1
ONEWIRE_NUM_BYTES_RX = 9
ONEWIRE_DATA_TX = [0xBE]
ONEWIRE_GO = 1

We can now read the 9 bytes from ONEWIRE_DATA_RX:

0x6A, 0x0A, 0x00, 0x00, 0x24, 0xAD, 0x2C, 0x22, 0x00

The 9 bytes contain the binary reading, a checksum, and some other information about the device. The devices used was set to 12-bit resolution so the conversion is 0.0625ºC/bit. The binary result is data[0] + data[1]*256. The binary temperature reading is 1*256 + 0x6A = 256 + 106 = 362. To convert that to ºC multiply by 0.0625. So the final temperature is 22.6 ºC.

1-Wire Registers

Name	Start Address	Type	Access	Default
ONEWIRE_DQ_DIONUM	5300	UINT16	R/W	0
ONEWIRE_DPU_DIONUM	5301	UINT16	R/W	0
ONEWIRE_OPTIONS	5302	UINT16	R/W	0
ONEWIRE_FUNCTION	5307	UINT16	R/W	0
ONEWIRE_NUM_BYTES_TX	5308	UINT16	R/W	0
ONEWIRE_NUM_BYTES_RX	5309	UINT16	R/W	0
ONEWIRE_GO	5310	UINT16	W	0
ONEWIRE_ROM_MATCH_H	5320	UINT32	R/W	0
ONEWIRE_ROM_MATCH_L	5322	UINT32	R/W	0
ONEWIRE_ROM_BRANCHS_FOUND_H	5332	UINT32	R	0
ONEWIRE_ROM_BRANCHS_FOUND_L	5334	UINT32	R	0
ONEWIRE_SEARCH_RESULT_H	5328	UINT32	R	0
ONEWIRE_SEARCH_RESULT_L	5330	UINT32	R	0
ONEWIRE_PATH_H	5324	UINT32	R/W	0
ONEWIRE_PATH_L	5326	UINT32	R/W	0
ONEWIRE_DATA_TX	5340	BYTE	R/W	0
ONEWIRE_DATA_RX	5370	BYTE	R/W	0

ONEWIRE_DQ_DIONUM
The data-line DIO number.

ONEWIRE_DPU_DIONUM
The dynamic pullup control DIO number.

ONEWIRE_OPTIONS
Controls advanced features. Value is a bitmask. bit 0: reserved, bit 1: reserved, bit 2: 1=DPU Enabled 0=DPU Disabled, bit 3: DPU Polarity 1=Active state is high, 2=Active state is low (Dynamic Pull-Up)

ONEWIRE_FUNCTION
Set the ROM function to use. 0xF0=Search, 0xCC=Skip, 0x55=Match, 0x33=Read.

ONEWIRE_NUM_BYTES_TX
Number of data bytes to be sent.

ONEWIRE_NUM_BYTES_RX
Number of data bytes to be received.

ONEWIRE_GO
Instructs the T7 to perform the configured 1-wire transaction.

ONEWIRE_ROM_MATCH_H
Upper 32-bits of the ROM to match.

ONEWIRE_ROM_MATCH_L
Lower 32-bits of the ROM to match.

ONEWIRE_ROM_BRANCHS_FOUND_H
Upper 32-bits of the branches detected during a search.

ONEWIRE_ROM_BRANCHS_FOUND_L

Lower 32-bits of the branches detected during a search.

ONEWIRE_SEARCH_RESULT_H
Upper 32-bits of the search result.

ONEWIRE_SEARCH_RESULT_L
Lower 32-bites of the search result.

ONEWIRE_PATH_H
Upper 32-bits of the path to take during a search.

ONEWIRE_PATH_L
Lower 32-bits of the path to take during a search.

ONEWIRE_DATA_TX
Data to be transmitted over the 1-wire bus.

ONEWIRE_DATA_RX
Data received over the 1-wire bus.

13.6 Asynchronous Serial

The T7 has UART functionality available that supports asynchronous (asynch) serial communication. The TX (transmit) and RX (receive) lines can appear on any digital I/O. Baud rates up to 38400 are supported, but the T7 processor is heavily loaded at that rate. Number of data bits, number of stop bits, and parity, are all controllable.

This asynchronous support is similar to RS-232, except that the logic is normal CMOS/TTL. Connection to an RS-232 device will require a converter chip such as the MAX233, which inverts the logic and shifts the voltage levels.

This serial link is not an alternative to the USB/Ethernet/WiFi connection. Rather, the host application will write/read data to/from the T7 over USB/Ethernet/WiFi, and the T7 communicates with some other device using the serial protocol. Using this serial protocol is considered an advanced topic. A good knowledge of the protocol is recommended, and a logic analyzer or oscilloscope might be needed for troubleshooting. Also consider that a better way to do RS-232 (or RS-485 or RS-422) communication is usually with a standard USB<=>RS-232 adapter/converter/dongle, so the user should have a particular reason to not use that and use a T7 instead.

Asynchronous Serial

Name	Start Address	Type	Access	Default
ASYNCH_ENABLE	5400	UINT16	R/W	0
ASYNCH_BAUD	5420	UINT32	R/W	0
ASYNCH_RX_DIONUM	5405	UINT16	R/W	0
ASYNCH_TX_DIONUM	5410	UINT16	R/W	0
ASYNCH_NUM_DATA_BITS	5415	UINT16	R/W	0
ASYNCH_RX_BUFFER_SIZE_BYTES	5430	UINT16	R/W	0
ASYNCH_NUM_BYTES_RX	5435	UINT16	R	0
ASYNCH_NUM_BYTES_TX	5440	UINT16	R/W	0
ASYNCH_TX_GO	5450	UINT16	W	0
ASYNCH_NUM_STOP_BITS	5455	UINT16	R/W	0
ASYNCH_PARITY	5460	UINT16	R/W	0
ASYNCH_NUM_PARITY_ERRORS	5465	UINT16	R/W	0
ASYNCH_DATA_TX	5490	UINT16	R/W	0
ASYNCH_DATA_RX	5495	UINT16	R/W	0

ASYNCH_ENABLE
1 = Turn on Asynch. Configures timing hardware, DIO lines and allocates the receiving buffer.

ASYNCH_BAUD
The symbol rate that will be used for communication. 9600 is typical. Up to 38400 works, but heavily loads the T7's processor.

ASYNCH_RX_DIONUM
The DIO line that will receive data. (RX)

ASYNCH_TX_DIONUM
The DIO line that will transmit data. (TX)

ASYNCH_NUM_DATA_BITS
The number of data bits per frame. 0-8, 0=8.

ASYNCH_RX_BUFFER_SIZE_BYTES
Number of bytes to use for the receiving buffer. Max is 2048. 0 = 200

ASYNCH_NUM_BYTES_RX
The number of data bytes that have been received.

ASYNCH_NUM_BYTES_TX
The number of bytes to be transmitted after writing to GO. Max is 256.

ASYNCH_TX_GO
Write a 1 to this register to initiate a transmission.

ASYNCH_NUM_STOP_BITS
The number of stop bits. 1 or 2. 0 = 1.

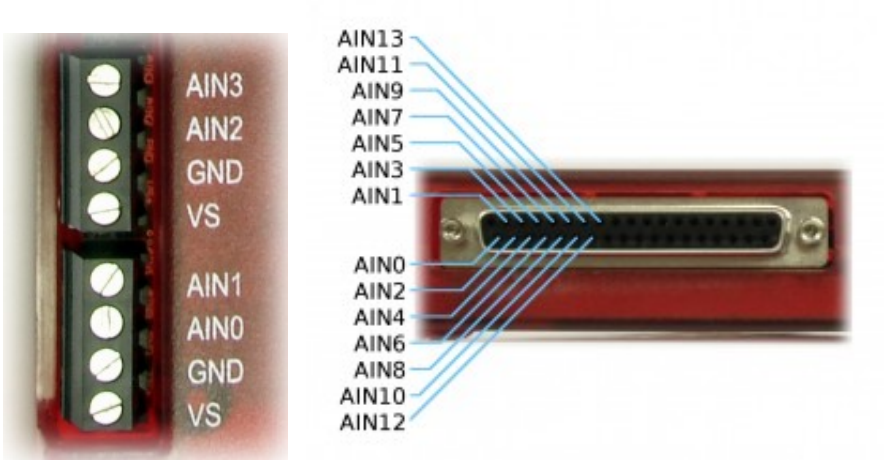
ASYNCH_PARITY
Parity setting: 0=none, 1=odd, 2=even.

ASYNCH_NUM_PARITY_ERRORS
The number of parity errors that have been detected. Cleared when UART is enabled. Can also be cleared by writing 0.

ASYNCH_DATA_TX
Write data to be transmitted here.

ASYNCH_DATA_RX
Read received data from here.

14.0 AIN



Analog Inputs: **14**

Voltage Ranges: **±10V, ±1V, ±0.1V, and ±0.01V**

T7 Max Resolution: **16-bit***

T7-Pro Max Resolution: **24-bit***

* See the Noise & Resolution Appendix for resolution details. For information regarding typical analog input connections, please see the Analog Signals App Note.

Analog Input Registers

Name	Start Address	Type	Access	Default
AIN#(0:13)	0	FLOAT32	R	
AIN#(0:13)_RANGE	40000	FLOAT32	R/W	0
AIN#(0:13)_NEGATIVE_CH	41000	UINT16	R/W	199
AIN#(0:13)_RESOLUTION_INDEX	41500	UINT16	R/W	0
AIN#(0:13)_SETTLING_US	42000	FLOAT32	R/W	0
AIN_ALL_RANGE	43900	FLOAT32	R/W	0

AIN_ALL_NEGATIVE_CH	43902	UINT16	R/W	199
AIN_ALL_RESOLUTION_INDEX	43903	UINT16	R/W	0
AIN_ALL_SETTLING_US	43904	FLOAT32	R/W	0

AIN#(0:13)
Returns the voltage of the specified analog input.

Names	Addresses
AIN0, AIN1, AIN2, Show All	0, 2, 4, Show All

AIN#(0:13)_RANGE
The range/span of each analog input. Write the highest expected input voltage.

Names	Addresses
AIN0_RANGE, AIN1_RANGE, AIN2_RANGE, Show All	40000, 40002, 40004, Show All

AIN#(0:13)_NEGATIVE_CH
Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Names	Addresses
AIN0_NEGATIVE_CH, AIN1_NEGATIVE_CH, AIN2_NEGATIVE_CH, Show All	41000, 41001, 41002, Show All

AIN#(0:13)_RESOLUTION_INDEX
The resolution index for each analog input. A larger resolution index generally results in lower noise and longer sample times.

Names	Addresses
AIN0_RESOLUTION_INDEX, AIN1_RESOLUTION_INDEX, AIN2_RESOLUTION_INDEX, Show All	41500, 41501, 41502, Show All

AIN#(0:13)_SETTLING_US
Settling time for command-response readings.

Names	Addresses
AIN0_SETTLING_US, AIN1_SETTLING_US, AIN2_SETTLING_US, Show All	42000, 42002, 42004, Show All

AIN_ALL_RANGE
A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

AIN_ALL_NEGATIVE_CH
A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

AIN_ALL_RESOLUTION_INDEX
A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

AIN_ALL_SETTLING_US
A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.
$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Some Examples

Analog Input Example: To read a voltage connected to AIN2, perform a read of AIN2 (or 4), and the result would be in the form of a floating point number, like 8.82332V.

Range Example: It is known that the voltage source connected to AIN1 will be 0 to 0.7V, so write 1.0 (or anything >0.1 and <=1.0) to AIN1_RANGE (40002), and the device will use the ±1V range.

Differential Analog Input Example: To do a differential reading on AIN2, you need to set AIN3 as its negative channel so that the measurement is AIN2-AIN3, rather then the default AIN2-GND (single-ended). Write a value of 3 to AIN2_NEGATIVE_CH (41002). To set it back to single-ended write a value of 199.

Resolution Index Example: Change the AIN1 resolution index to 5 by writing a value of 5 to AIN1_RESOLUTION_INDEX (41501).

Settling Example: Change the settling time of AIN3 to 500uS by writing a value of 500 to AIN3_SETTLING_US (42006), although we recommend a value of 0 which corresponds to automatic settling.

Extra Details

The analog inputs are not artificially pulled to 0.0 volts, as that would reduce the input impedance, so readings obtained from floating channels will generally not be 0.0 volts. The readings from floating channels depend on adjacent channels and sample rate and have little meaning. See related floating input application note.

Notice that the addresses for AIN#, AIN#_RANGE, and AIN#_SETTLING_US increment in steps of 2. That is because they are FLOAT32 and thus each needs 2 16-bit registers. AIN#_NEGATIVE_CH and AIN#_RESOLUTION_INDEX are UNIT16 and thus just use 1 16-bit register each, so their addresses step in increments of 1.

The **AIN#(0:13)_RANGE** parameter is actually controlling the gain of the internal instrumentation amplifier. The in-amp supports gains of x1, x10, x100, and x1000. If you set range=10, you get gain=x1, and the analog input range is ±10 volts. If you set range=1, you get gain=x10, and the analog input range is ±1 volts. Note that the device knows what the internal gain is set to and adjusts the return values to give the voltage at the input terminals, so if you connect a 0.8 volt signal to the input terminals, it will be amplified to 8.0 volts before being digitized, but the reading you get back will be 0.8 volts. Write range=10 to get a range of ±10V (default), range=1 to get a range of ±1V, range=0.1 to get a range of ±0.1V, or range=0.01 to get a range of ±0.01V. If you write a value in between the valid ranges, the larger range will be used.

The **AIN#(0:13)_NEGATIVE_CH** parameter pertains to differential readings. On the T7, differential channels are adjacent even/odd pairs only, such as AIN2-AIN3. Thus the positive channel must be even and the negative channel must be +1. Only an even channel can have an associated negative channel, so you will never write to odd channels of this register (e.g. never write to AIN3_NEGATIVE_CH/41003). Channel numbers in the extended range (above AIN15), are connected to AIN0-AIN13 and those dictate the even/odd rule, not the extended channel numbers (see the Mux80 datasheet).

The **AIN#(0:13)_RESOLUTION_INDEX** parameter affects the ADC. A higher Resolution_Index results in lower noise and thus higher effective & noise-free resolution, with the tradeoff of longer sample times. See the Noise and Resolution Appendix. The value passed for Resolution_Index is from 0-8, where 0 corresponds to default, 1 is roughly 16-bit resolution (RMS or effective), and 8 is roughly 19-bit resolution. The T7-Pro has additional Resolution_Index settings 9-12 that use the alternate high-resolution converter (24-bit sigma-delta) and correspond to roughly 19-bit to 22-bit resolution. For command-response readings, the default value of 0 corresponds to Resolution_Index=8 on a T7 and Resolution_Index=9 on a T7-Pro. For stream readings the default of 0 corresponds to Resolution_Index=1. For typical noise levels and sample times at different combinations of resolution index and gain see Appendix A-3-1 Noise and Resolution. For general discussion on the meaning of resolution see the Noise and Resolution App Note.

The **AIN#(0:13)_SETTLING_US** parameter is the time from a step change in the input signal to when the signal is sampled by the ADC, measured in microseconds. A step change in this case is caused when the internal multiplexers change from one channel to another. In general, more settling time is required as gain and resolution are increased. The default “auto” settling time ensures that the device meets specifications at any gain and resolution for source impedance up to at least 1000 ohms. This parameter applies to command/response mode and AIN EF. Stream mode has its own settling parameter. The timings in Appendix A are measured with “auto” settling.

14.1 AIN Extended Features

Analog extended features (AIN-EF) are used for advanced operations involving analog inputs. Possible features:

- **Linear Scaling:** Apply a simple slope and offset to convert a voltage into something else, like pressure, force, or temperature.
- **FlexRMS:** Calculate RMS voltage. Flexible sample rate, number of samples, and hysteresis parameters. Also returns frequency/period.
- **Thermocouples:** Perform the math to handle cold junction compensation (CJC) and voltage to temperature conversion. Firmware automatically handles any complicated math.
- **RTDs:** Automatically calculate a resistance value from an RTD. Can use either a constant current source, known excitation voltage, or secondary measurements across a shunt resistor.

AIN-EF is only supported in command-response mode ... not stream mode.

Configuration:

Before an AIN_EF can be used a few settings need to be written. The desired operation is selected by writing the associated index number to AIN#(0:13)_EF_INDEX. Depending on the selected index several configuration registers can be written. The details of the configuration registers can be found in the associated index section.

Analog input settings such as range, resolution, settling, and negative channel, are configured through the normal AIN registers.

Reading:

When READ_A is read from the LabJack it will read from analog inputs, run calculations, and then return the result. If the selected index produces more than one result they will be saved so that they can be read later. Reading from result registers other than A, read from the saved values and do not initiate a new reading.

List of index values:

- **0:** None (disabled)
- **1:** Slope Offset
- **11:** FlexRMS
- **20:** Thermocouple type E
- **21:** Thermocouple type J
- **22:** Thermocouple type K
- **23:** Thermocouple type R
- **24:** Thermocouple type T
- **24:** Thermocouple type T
- **24:** Thermocouple type T
- **40:** RTD PT100
- **41:** RTD PT500
- **42:** RTD PT1000

Analog Extended Features

Name	Start Address	Type	Access	Default
AIN#(0:13)_EF_READ_A	7000	FLOAT32	R	0
AIN#(0:13)_EF_READ_B	7300	FLOAT32	R/W	0
AIN#(0:13)_EF_READ_C	7600	FLOAT32	R/W	0
AIN#(0:13)_EF_READ_D	7900	FLOAT32	R	0
AIN#(0:13)_EF_INDEX	9000	UINT32	R/W	0
AIN_ALL_EF_INDEX	43906	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_A	9300	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_B	9600	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_C	9900	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_D	10200	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_E	10500	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_F	10800	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_G	11100	FLOAT32	R/W	0

AIN#(0:13)_EF_READ_A

Names	Addresses
AIN0_EF_READ_A, AIN1_EF_READ_A, AIN2_EF_READ_A, Show All	7000, 7002, 7004, Show All

AIN#(0:13)_EF_READ_B

Names	Addresses
AIN0_EF_READ_B, AIN1_EF_READ_B, AIN2_EF_READ_B, Show All	7300, 7302, 7304, Show All

AIN#(0:13)_EF_READ_C

Names	Addresses
AIN0_EF_READ_C, AIN1_EF_READ_C, AIN2_EF_READ_C, Show All	7600, 7602, 7604, Show All

AIN#(0:13)_EF_READ_D

Names	Addresses
AIN0_EF_READ_D, AIN1_EF_READ_D, AIN2_EF_READ_D, Show All	7900, 7902, 7904, Show All

AIN#(0:13)_EF_INDEX

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled), 1=Slope/Offset, 11=FlexRMS, 20=Thermocouple type E, 21=Thermocouple type J, 22=Thermocouple type K, 23=Thermocouple type R, 24=Thermocouple type T, 40=RTD model PT100, 41=RTD model PT500, 42=RTD model PT1000.

Names	Addresses
AIN0_EF_INDEX, AIN1_EF_INDEX, AIN2_EF_INDEX, Show All	9000, 9002, 9004, Show All

AIN_ALL_EF_INDEX

Write 0 to deactivate AIN_EF on all AINs. No other values may be written to this register. Reads will return the AIN_EF index if all 128 AINs are set to the same value. If values are not the same retuns 0xFFFF (65535).

AIN#(0:13)_EF_CONFIG_A

Names	Addresses

AIN0_EF_CONFIG_A, AIN1_EF_CONFIG_A, AIN2_EF_CONFIG_A, Show All	9300, 9302, 9304, Show All

AIN#(0:13)_EF_CONFIG_B

Names	Addresses
AIN0_EF_CONFIG_B, AIN1_EF_CONFIG_B, AIN2_EF_CONFIG_B, Show All	9600, 9602, 9604, Show All

AIN#(0:13)_EF_CONFIG_C

Names	Addresses
AIN0_EF_CONFIG_C, AIN1_EF_CONFIG_C, AIN2_EF_CONFIG_C, Show All	9900, 9902, 9904, Show All

AIN#(0:13)_EF_CONFIG_D

Names	Addresses
AIN0_EF_CONFIG_D, AIN1_EF_CONFIG_D, AIN2_EF_CONFIG_D, Show All	10200, 10202, 10204, Show All

AIN#(0:13)_EF_CONFIG_E

Names	Addresses
AIN0_EF_CONFIG_E, AIN1_EF_CONFIG_E, AIN2_EF_CONFIG_E, Show All	10500, 10502, 10504, Show All

AIN#(0:13)_EF_CONFIG_F

Names	Addresses
AIN0_EF_CONFIG_F, AIN1_EF_CONFIG_F, AIN2_EF_CONFIG_F, Show All	10800, 10802, 10804, Show All

AIN#(0:13)_EF_CONFIG_G

Names	Addresses
AIN0_EF_CONFIG_G, AIN1_EF_CONFIG_G, AIN2_EF_CONFIG_G, Show All	11100, 11102, 11104, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

14.1.1 Thermocouples

Index: **20-24**

The AIN extended features system can automatically perform the necessary calculations for Type E, J, K, R, and T thermocouples.

Thermocouple modes read an analog input connected to a thermocouple and a second specified AIN connected to a CJC sensor. The CJC slope and offset are used to compute the CJC temperature then the thermocouple temperature is calculated. The on-board temperature sensor is the default CJC channel.

The thermocouple feature will use the normal analog input settings for negative channel, resolution-index, and settling. For example, if you want a differential reading of AIN0-AIN1, set AIN0_NEGATIVE_CH = 1.

If the range of the applicable channel is set to the default ±10 volts, the thermocouple feature will automatically use the ±0.1 volt range, otherwise the specified range will be used.

CJC calculations are always done in degrees K, regardless of whether Config-A is used to change the output units. Config-D and Config-E should be used as needed to scale the CJ sensor reading to degrees K.

Thermocouple feature index values:
20: Thermocouple type E
21: Thermocouple type J
22: Thermocouple type K
23: Thermocouple type R
24: Thermocouple type T

Configuration Registers:
CONFIG_A: TC_Options – Bitmask containing additional options. Bits 0 and 1 change the units of the output of the calculations (READ_A & READ_C). The default is kelvin. Bit 0: 1 = Report in ºC, Bit 1: 1 = Report in ºF.
CONFIG_B: CJC_ModbusAddress – This is the modbus address that will be read to acquire the CJC reading. The default is 60052;

TEMPERATURE_DEVICE_K.
CONFIG_D: CJC_Slope – Slope to be applied to the CJC reading. This value is always °K/volt regardless of CONFIG_A setting, so for the internal sensor will nominally be 1.00 and for an LM34 will be 55.56. Default is 1.0.
CONFIG_E: CJC_Offset – Offset to be applied to the CJC reading. This value is always °K regardless of CONFIG_A setting, so for the internal sensor will nominally be 0.0 and for an LM34 will be 255.37. Default it 0.0.

Result Registers:
READ_A: Final calculated temperature.
READ_B: Measured thermocouple voltage.
READ_C: CJC temperature.
READ_D: Thermocouple voltage calculated for CJC temperature.

Example:
Assume a type K thermocouple is connected to AIN3/GND. First write some values to configure:

```
AIN3_EF_INDEX = 22              // feature index for type K thermocouple
AIN3_EF_CONFIG_B = 60052    // CJC source, address for device temperature sensor
AIN3_EF_CONFIG_D = 1.0       // slope for CJC reading
AIN3_EF_CONFIG_E = 0.0       // offset for CJC reading
```

Now each read of AIN3_EF_READ_A will cause a new measurement to happen and return the temperature of the remote end of the thermocouple. If the remote end is at room temperature you should read about 298 degrees K.

For more information see the Thermocouples Application Note and its T7-specific child page.

14.1.2 Offset and Slope

Index: **1**

The T7 AIN extended feature system can automatically add a slope and a offset to analog readings, the results of which can be read through the AIN EF registers.

Configuration Registers:
CONFIG_D: Custom slope to be applied to the analog voltage reading. Default is 1.00.
CONFIG_E: Custom Offset to be applied to the analog voltage reading. Default is 0.00.

Result Registers:
READ_A: returns measured volts * slope + offset.

Example:
First write some values to configure the AIN-EF for AIN3:

```
AIN3_EF_INDEX = 1              // feature index for type K thermocouple
AIN3_EF_CONFIG_D = 2.0       // slope
AIN3_EF_CONFIG_E = -0.5      // offset
```

Now each read of AIN3_EF_READ_A will cause a new measurement to happen and return (2.0 * volts) - 0.5.

14.1.3 RTD

BETA

Index: **40-42** AIN Extended Features can compute the temperatures of a resistance temperature detector.

First, the resistance of the RTD is determined, then the rational polynomial technique is used to calculate the temperature that resistance corresponds to. There are 2 general supported methods to determine the resistance of the RTD: current source or voltage divider.

Current source: Resistance is calculated as R = V / I. V is the voltage reading from this AIN-EF channel ("#", configure this channel as differential if needed), and I is the value of the current source. This feature can read the stored factory calibration value of the 200UA or 10UA source, can use a specified constant value (in amps) for the current source, or can measure the voltage across a fixed shunt

resistance (which is in series with the RTD) to determine current. For the latter you must specify the resistance of the shunt (in ohms) and the AIN channel to use to measure the voltage across the shunt (configure this channel as differential if needed).

Voltage divider: Resistance is calculated as R = (Vexc - V) * R2 / V. V is the voltage reading from this AIN-EF channel ("#", configure this channel as differential if needed). R2 is the fixed voltage divider resistor that goes from the RTD to GND and you must specify the value in ohms. Vexc is the excitation voltage for the voltage divider. You either specify this value in volts or it is measured on a specified AIN channel.

RTD feature index values:
40: PT100
41: PT500
42: PT1000

Configuration Registers:
CONFIG_A: RTD_Options – Bitmask containing additional options. Bits 0 and 1 change the units of the output of the calculations. The default is kelvin. Bit 0: 1 = Report in ºC, Bit 1: 1 = Report in ºF.
CONFIG_B: Excitation Type – This option tells the T7 how the RTD is connected. There are several options:

- 0: Current source, 200 µA, use factory cal value.
- 1: Current source, 10 µA, use factory cal value.
- 2: Current source, specify amps in CONFIG_?.
- 3: Shunt resistor, specify Rshunt ohms and AIN# to measure shunt voltage.
- 4: Voltage source, specify R2 ohms and Vexc volts.
- 5: Voltage source, specify R2 ohms and AIN# to measure Vexc voltage.

CONFIG_C: Excitation AIN # - Channel number of the AIN line used to measure the RTD's excitation.
CONFIG_D: Reserved
CONFIG_E: Excitation detail - Volts
CONFIG_F: Excitation detail - Ohms
CONFIG_G: Excitation detail - Amps

Result Registers:
READ_A: Final calculated temperature.
READ_B: Resistance of the RTD.
READ_C: Voltage across the RTD.
READ_D: Current through the RTD.

14.1.4 FlexRMS

BETA

Index: **11**

Min Firmware: **1.0124**

This analog input extended feature (AIN-EF) determines the true RMS voltage of a signal on an analog input. It also returns the peak-to-peak voltage, DC offset (average voltage), and period (1/frequency) of the analog waveform.

This feature uses stream-burst internally to acquire a specific number of scans at a hardware controlled scan rate. Set the scan rate to get the desired time resolution. Set the number of scans to make sure you get at least 2 falling or 2 rising zero crossings. A zero crossing is where the waveform crosses the average value.

After acquiring the entire data set, we calculate an initial average of the entire data set, and then must find 1 cycle (rising-to-rising or falling-to-falling) of the waveform, which is done as follows:

-Wait for value > (average+hysteresis) or value < (average-hysteresis).

If ">" find next 2 falling edge zero crossings:

-Wait for value <= average.
-1st falling edge zero crossing.
-Wait for value < (average-hysteresis).
-Wait for value > (average+hysteresis).
-Wait for value <= average.

-2nd falling edge zero crossing.

If "<" find next 2 rising edge zero crossings:

-Wait for value >= average.
-1st rising edge zero crossing.
-Wait for value > (average+hysteresis).
-Wait for value < (average-hysteresis).
-Wait for value >= average.
-2nd rising edge zero crossing.

We can now note the period of the waveform as the time between the 2 falling or 2 rising zero crossings. Finally, we use the 1 cycle of data to calculate RMS & peak-to-peak, and the final value for average.

RMS is calculated as ((Σ(Point^2)) / numPoints) ^0.5, and thus provides true RMS regardless of waveform shape.

Configuration Registers:

CONFIG_A: NumScans (default=200, max=16384)
CONFIG_B: Hysteresis (default=100, 16-bit counts)
CONFIG_D: ScanRate (default=6000)

Result Registers:
READ_A: RMS Voltage
READ_B: Peak-to-Peak Voltage
READ_C: DC Offset Voltage (Average)
READ_D: Period (Seconds)

Example:

...

AIN#_EF_INDEX = 11

14.1.4 FlexRMS

14.2 Special Channels

The T7 has special channels to allow for various analog input features. All the analog inputs (0-13) can be used for single-ended readings. For differential readings, channels are grouped into pairs, the positive channel is an even number, and the negative channel is odd. Only adjacent channel numbers can be used as differential, e.g. 0 and 1 form a pair, but 0 and 3 cannot form a pair.

Differential channel pairs

Differential Pair	Positive AIN	Negative AIN
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	12	13

Other unique analog channels

AIN	Function
14	Internal Temperature Sensor (volts)
15,199	Ground (GND)

Special AINs

Name	Start Address	Type	Access	Default
AIN#(14:15)	28	FLOAT32	R	

AIN#(14:15)
Returns the voltage of the specified analog input.

Names	Addresses
AIN14, AIN15	28, 30

It is possible to read GND directly via AIN15, or AIN199. Read more on the temperature sensor in temperature sensor section.

14.3 Extended Channels

The Mux80 is a ready-made analog input expansion board which adds 80 analog inputs when used in conjunction with a T7. The extended channels can be read using the following registers. For details about the physical mapping of pins, see the Mux80 Datasheet.

For differential extended channels, the positive channel must map to an even channel from 0-12, and the negative channel must map to the odd channel 1 higher (i.e. 1-13). That means that for extended channel numbers the negative channel must be 8 higher than the positive channel. For example, a valid differential extended channel pair would be Ch+ = AIN102 and Ch- = AIN110, since AIN102 maps to AIN6 and AIN110 maps to AIN7. For more information on differential extended channels, see the Mux80 Datasheet.

Note that when using the Mux80 board, the T7's MIO(0-2) lines are consumed for multiplexer signaling.

Mux80 Extended Channels

Name	Start Address	Type	Access	Default
AIN#(48:127)	96	FLOAT32	R	

AIN#(48:127)
Returns the voltage of the specified analog input.

Names	Addresses
AIN48, AIN49, AIN50, Show All	96, 98, 100, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

15.0 DAC

Output: **0V to 5V**

Resolution: **12-bit**

Source Impedance: **50 ohms**

Max Output Current: **20mA***

*At max current the output voltage will be lower than configured, see Appendix A-4 for details.

DAC Registers

Name	Start Address	Type	Access	Default
DAC#(0:1)	1000	FLOAT32	R/W	

DAC#(0:1)
Pass a voltage for the specified analog output.

Names	Addresses

DAC0, DAC1	1000, 1002

Overview

There are two DACs (digital-to-analog converters or analog outputs) on the T7. Each DAC can be set to a voltage between about 0.01 and 5 volts with 12-bits of resolution.

For electrical specifications, See Appendix A-4.

Although the DAC values are based on an absolute reference voltage, and not the supply voltage, the DAC output buffers are powered internally by VS and thus the maximum output voltage is limited to slightly less than VS.

The DACs appear both on the screw terminals and on the DB37 connector. These connections are electrically the same, and the user must exercise caution only to use one connection or the other, and not create a short circuit.

Power-up Defaults

The power-up condition of the DACs can be configured by the user. From the factory, the DACs default to enabled at minimum voltage (~0 volts). Note that even if the power-up default for a line is changed to a different voltage or disabled, there is a delay of about 100 ms at power-up where the DACs are in the factory default condition.

Protection

The analog outputs can withstand a continuous short-circuit to ground, even when set at maximum output.

Voltage should never be applied to the analog outputs, as they are voltage sources themselves. In the event that a voltage is accidentally applied to either analog output, they do have protection against transient events such as ESD (electrostatic discharge) and continuous overvoltage (or undervoltage) of a few volts.

Increase Output to ±10V

There is an accessory available from LabJack called the LJTick-DAC that provides a pair of 14-bit analog outputs with a range of ±10 volts. The LJTick-DAC plugs into any digital I/O block, and thus up to 10 of these can be used per T7 to add 20 analog outputs.

16.0 DB37

Number of Pins: **37**

Screw type: **#4-40**

Contacts: **Gold-coated**

Form factor: **D-Sub**

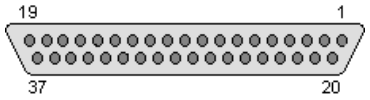
This high-density connector provides access to the T7 features that are not available on the screw terminal edge of the unit. It brings out analog inputs (AIN), analog outputs (DAC), digital I/O (FIO, MIO), and other signals.

The CB37 is a connector board that provides convenient screw-terminals for the DB37 lines, but the CB37 is not required to access I/O on the DB37. Any method you see fit can be used to access the DB37 lines.

Some signals appear on both the DB37 connector and screw terminals, so care must be taken to avoid contention. For such signals, only connect to one location, not both. Signals shared between T7 screw terminals and the DB37 are denoted in **bold**.

Pinout

DB37 Pinouts						
1	**GND**	14	AIN9	27	**Vs**	
2	**200uA**	15	AIN7	28	Vm+	
3	FIO6	16	AIN5	29	**DAC1**	
4	FIO4	17	**AIN3**	30	**GND**	
5	**FIO2**	18	**AIN1**	31	AIN12	
6	**FIO0**	19	**GND**	32	AIN10	
7	MIO1	20	**10uA**	33	AIN8	
8	**GND**	21	FIO7	34	AIN6	
9	Vm-	22	FIO5	35	AIN4	
10	**GND**	23	**FIO3**	36	**AIN2**	
11	**DAC0**	24	**FIO1**	37	**AIN0**	
12	AIN13	25	MIO0			
13	AIN11	26	MIO2			
DB37 Connector Pinouts						



The above image shows standard DB37 pin numbers looking into the female connector on the T7.

VS, GND, FIO/MIO, AIN, DAC, 200UA/10UA

Descriptions of these can be found in their related sections of this datasheet.

VM+/VM-

Vm+/Vm- are bipolar power supplies intended to power external multiplexer ICs such as the DG408 from Intersil. The multiplexers can only pass signals within their power supply range, so Vm+/Vm- can be used to pass bipolar signals. Nominal voltage is ±13 volts at no load and ±12 volts at 2.5 mA. Both lines have a 100 ohm source impedance, and are designed to provide 2.5 mA or less. This is the same voltage supply used internally by the T7 to bias the analog input amplifier and multiplexers. If this supply is loaded more than 2.5 mA, the voltage can droop to the point that the maximum analog input range is reduced. If this supply is severely overloaded (e.g. short circuited), then damage could eventually occur. If Vm+/Vm- are used to power multiplexers, series diodes are recommended as shown in Figure 9 of the Intersil DG408 datasheet. Not so much to protect the mux chips, but to prevent current from going back into Vm+/Vm-. Use Schottky diodes to minimize voltage drop.

CB37 Terminal Board

The CB37 terminal board from LabJack connects to the DB37 connector and provides convenient screw terminal access to all lines. The CB37 is designed to connect directly to the DB37, but can also connect via a 37-line 1:1 male-female cable.

When using the analog connections on the CB37, the effect of ground currents should be considered, particularly when a cable is used and substantial current is sourced/sunk through the CB37 terminals. When any sizeable cable lengths are involved, a good practice is to separate current carrying ground from ADC reference ground. An easy way to do this on the CB37 is to use GND as the current source/sink, and use AGND as the reference ground. This works well for passive sensors (no power supply), such as a thermocouple, where the only ground current is the return of the input bias current of the analog input.

EB37 Experiment Board

The EB37 experiment board connects to the DB37 connector and provides convenient screw terminal access. Also provided is a solderless breadboard and useful power supplies. The EB37 is designed to connect directly to the DB37, but can also connect via a 37-line 1:1 male-female cable.

OEM

The OEM T7 has a separate header location to bring out the same connections as the DB37 connector. This OEM header location is labeled J3. The J3 holes are always present, but are obstructed when the DB37 connector is installed. Find the pinout, and other OEM information for J3 in OEM Versions.

17.0 DB15

Number of Pins: **15**

Screw type: **#4-40**

Contacts: **Gold-coated**

Form factor: **D-Sub**

The DB15 connector brings out 12 additional digital I/O. It has the potential to be used as an expansion bus, where the 8 EIO are data lines and the 4 CIO are control lines. EIO0-CIO3 can also be addressed as DIO8-DIO19.

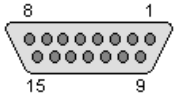
```
EIO0-EIO7  aka DIO8-DIO15
CIO0-CIO3  aka DIO16-DIO19
```

The CB15 is a connector board that provides convenient screw-terminals for the DB15 lines, but the CB15 is not required to access I/O on the DB15. Any method you see fit can be used to access the DB15 lines.

These 12 channels include an internal series resistor that provides overvoltage/short-circuit protection. These series resistors also limit the ability of these lines to sink or source current. Refer to the specifications in Appendix A-2. All digital I/O on the U6 have 3 possible states: input, output-high, or output-low. Each bit of I/O can be configured individually. When configured as an input, a bit has a ~100 kΩ pull-up resistor to 3.3 volts. When configured as output-high, a bit is connected to the internal 3.3 volt supply (through a series resistor). When configured as output-low, a bit is connected to GND (through a series resistor).

DB15 Pinouts				
1	Vs	9	CIO0	
2	CIO1	10	CIO2	
3	CIO3	11	GND	
4	EIO0	12	EIO1	
5	EIO2	13	EIO3	
6	EIO4	14	EIO5	
7	EIO6	15	EIO7	
8	GND			

DB15 Connector Pinouts



The above image shows standard DB15 pin numbers looking into the female connector on the T7.

CB15

The CB15 terminal board connects to the DB15 connector. It provides convenient screw terminal access to the 12 digital I/O available on the DB15 connector. The CB15 is designed to connect directly to the DB15, or can connect via a standard 15-line 1:1 male-female DB15 cable.

RB12

The RB12 relay board provides a convenient interface for the T7 to industry standard digital I/O modules, allowing electricians, engineers, and other qualified individuals, to interface a LabJack with high voltages/currents. The RB12 relay board connects to the DB15 connector on the LabJack, using the 12 EIO/CIO lines to control up to 12 I/O modules. Output or input types of digital I/O modules can be used. The RB12 is designed to accept G4 series digital I/O modules from Opto22, and compatible modules from other manufacturers such as the G5 series from Grayhill. Output modules are available with voltage ratings up to 200 VDC or 280 VAC, and current ratings up to 3.5 amps.

OEM

The OEM T7 has a separate header location to bring out the same connections as the DB15 connector. This OEM header location is labeled J2. The J2 holes are always present, but are obstructed when the DB15 connector is installed. Find the pinout, and other OEM information for J2 in OEM Versions.

18.0 Internal Temp Sensor

Sensor Range: **-50°C to 150°C**

T7 Operating Range: **-40°C to 85°C**

Accuracy (20°C to 40°C): **±1.5°C***

Accuracy (-20°C to 50°C): **±2.0°C***

Accuracy (-45°C to 85°C): **±3.5°C***

*Accuracy of measuring device temperature, which is typically warmer than ambient air temperature. Includes error from LM94021 specifications and error due to linear equation fit.

The T7 has an LM94021 temperature sensor connected to internal analog input channel 14 (AIN14). The sensor is physically located on the bottom of the PCB between the AIN0/1 and AIN2/3 screw-terminals. A reading from AIN14 returns volts, which can be converted to device temperature using the formula volts*-92.6 + 467.6. Alternatively, read the temperature in degrees Kelvin using the registers TEMPERATURE_AIR_K and TEMPERATURE_DEVICE_K.

Internal Temp Sensor

Name	Start Address	Type	Access	Default
TEMPERATURE_AIR_K	60050	FLOAT32	R	
TEMPERATURE_DEVICE_K	60052	FLOAT32	R	

TEMPERATURE_AIR_K
Returns the estimated ambient air temperature just outside a T7 in its red plastic enclosure. This register is equal to TEMPERATURE_DEVICE_K - 4.3. If Ethernet and/or WiFi is enabled, subtract an extra 0.6 for each.

TEMPERATURE_DEVICE_K
Takes a reading from AIN14 using range=+/-10V and resolution=8, and applies the formula Volts*-92.6+467.6 to return degrees K. AIN14 is internally connected to an LM94021 (U24) with GS=10 which is physically located on the bottom of the PCB between the AIN0/1 and AIN2/3 screw-terminals.

Offset considerations

The unadjusted sensor reading best reflects the temperature of the device inside the enclosure and the temperature of the AIN0-3 screw-terminals. This is what you get from TEMPERATURE_DEVICE_K in degrees Kelvin.

TEMPERATURE_AIR_K is an estimate of the ambient air temperature outside the device. It is calculated depending on whether Ethernet and/or WiFi is enabled as follows:

USB	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.3
USB & Ethernet	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.9
USB & WiFi	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.9
USB & Ethernet & WiFi	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 5.5

These offsets were determined from measurements with the enclosure on and in still air. We noted that the time constant was about 12 minutes, meaning that 12 minutes after a step change you are 63% of the way to the new value.

Note on thermocouples

The value from register TEMPERATURE_DEVICE_K best reflects the temperature of the built-in screw-terminals AIN0-AIN3, so use that for cold junction compensation (CJC) if thermocouples are connected there.

The internal sensor has a specified accuracy of ±2.0 °C across the range of -20 to +50 °C. Allowing for a slight difference between the sensor temperature and the temperature of the screw-terminals, expect the returned value minus 3 °C to reflect the temperature of the built-in screw-terminals with an accuracy of ±2.5 °C.

If thermocouples are connected to the CB37, you want to know the temperature of the screw-terminals on the CB37. The CB37 is typically at the same temperature as ambient air, so use the value from register TEMPERATURE_AIR_K for CJC. Better yet, add a sensor such as the LM34CAZ to an unused analog input on the CB37 to measure the actual temperature of the CB37.

19.0 RTC (T7-Pro only)

The T7-Pro has a battery-backed RTC (real-time clock) which is useful for assigning timestamps to data that is stored on the µSD card during scripting operations. Particularly in situations where the device could experience power failure or other reboots, and does not have a communication connection that can be used to determine real-time after reboot. The system time is stored in seconds since 1970, also known as the Epoch & Unix timestamp.

Remarks

As of this writing, the T7-Pro ships with a 2GB SD card, RTC, and battery/battery-holder installed. The T7 has none of these, but does have the socket installed to hold a micro SD card.

The T7-Pro RTC can be read a few different ways. Read the system time in seconds with address 61500, or get a simple calendar time representation by reading six consecutive addresses, starting with address 61510. Change the system time by writing a new timestamp(in seconds) to address 61504. To time events faster than 1 second apart, it is possible to read the CORE_TIMER, address 61520, and see how it changes from second to second. To access the core timer value in Lua scripts, use the LJ.Tick() function.

RTC Registers

Name	Start Address	Type	Access	Default
RTC_TIME_S	61500	UINT32	R	
RTC_SET_TIME_S	61504	UINT32	W	
RTC_SET_TIME_SNTP	61506	UINT32	W	
RTC_TIME_CALENDAR	61510	UINT16	R	

RTC_TIME_S
Read the current time on the RTC in seconds since Jan, 1970, aka Epoch or Unix time.

RTC_SET_TIME_S
Write a new timestamp to the RTC in seconds since Jan, 1970, aka Epoch or Unix timestamp.

RTC_SET_TIME_SNTP
Write any value to instruct the T7 to update its clock from a SNTP server.

RTC_TIME_CALENDAR
Read six consecutive addresses of type UINT16, starting with this address. The result will be in year, month, day, hour, minute, second calendar format. i.e. [2014, 10, 21, 18, 55, 35]

Examples

[Lua] Read the value of the RTC in a Lua script

```
table = {}
table[1] = 0    --year
table[2] = 0    --month
table[3] = 0    --day
table[4] = 0    --hour
table[5] = 0    --minute
table[6] = 0    --second

table, error = MB.RA(61510, 0, 6)
print(string.format("%04d/%02d/%02d %02d:%02d.%02d", table[1], table[2], table[3], table[4], table[5], table[6]))

>> 2014/10/15 18:55.22
```

[C/C++] Read the value of the RTC in C/C++

```
int LJMError;
double newValue;
LJMError = LJM_eReadAddress(handle, 61500, 1, &newValue);
printf(newValue);
//returned value of 1413398998 would correspond with with Wed, 15 Oct 2014 18:49:58 GMT
```

19.0 RTC

20.0 Internal Flash

The T7 has a flash built-in flash memory chip, and a micro SD card. The built-in flash is used to store calibration constants, among other things. The following registers are used to access built-in flash memory. For information about the SD card, see the related section.

Flash Memory

Name	Start Address	Type	Access	Default
INTERNAL_FLASH_KEY	61800	UINT32	R/W	

INTERNAL_FLASH_READ_POINTER	61810	UINT32	R/W
INTERNAL_FLASH_READ	61812	UINT32	R
INTERNAL_FLASH_WRITE_POINTER	61830	UINT32	R/W
INTERNAL_FLASH_WRITE	61832	UINT32	W
INTERNAL_FLASH_ERASE	61820	UINT32	W

INTERNAL_FLASH_KEY
Sets the region of internal flash to which access is allowed.

INTERNAL_FLASH_READ_POINTER
The address in internal flash that reads will start from.

INTERNAL_FLASH_READ
Data read from internal flash.

INTERNAL_FLASH_WRITE_POINTER
Address in internal flash where writes will begin.

INTERNAL_FLASH_WRITE
Data written here will be written to internal flash.

INTERNAL_FLASH_ERASE
Erases a 4k section of internal flash starting at the specified address.

61812: Read 1-512 registers starting from this address to get the data. Flash is read in 32-bit chunks, so you must read an even number of registers. You can only read multiple registers starting from this Modbus address ... you can't read 61812, then read 61814, and so on. The number of registers you can read at once might be further limited by the maximum packet size of the particular interface ... if you don't want to worry about that just stick to 13 values (26 registers) or less per read.

For example: To read 8 floats out of memory, starting at external flash address 3948544, initialize the read pointer (Modbus address 61810) to a value of 3948544 using eWriteAddress(), then read Modbus addresses starting at address 61812 using eReadAddresses(). The read pointer (address 61810) does not automatically increment.

Calibration Constants

The T7 automatically returns calibrated readings, so most people should not concern themselves with this section.

If the factory applied calibration constants are of interest, they are stored on internal memory and can be accessed at any time through the use of the Modbus registers listed in the table above.

The cal constants begin at memory address 0x3C4000, or in decimal format 3948544. The structure(location) of each calibration value can be seen in the code snippet below.

Follow the above example to read out the first 8 values: PSlope, NSlope, Center, Offset (HS Gain= x1), and PSlope, NSlope, Center, Offset (HS Gain= x10).

```
typedef struct{
 float PSlope;
 float NSlope;
 float Center;
 float Offset;
}Cal_Set;

typedef struct{
 Cal_Set HS[4];
 Cal_Set HR[4];

 struct{
  float Slope;
  float Offset;
 }DAC[2];

 float Temp_Slope;
 float Temp_Offset;

 float ISource_10u;
 float ISource_200u;

 float I_Bias;
}Device_Calibration;
```

The full size of the calibration section is 164 bytes, or 41 floats.

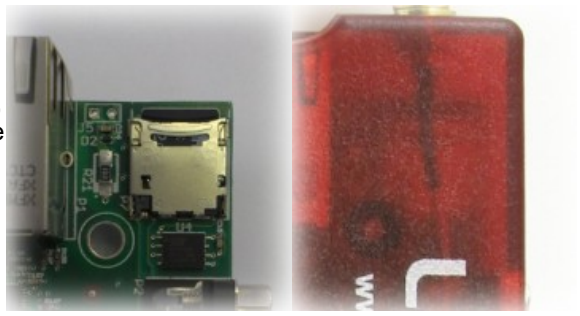
The reason that there are 'Cal_Set's for each High Speed 'HS' and High Resolution 'HR', is that there are 2 analog converters on a T7-Pro. A standard T7 uses only the High Speed analog converter, so only the HS[4] calibration values will be populated with valid information. A T7-Pro will have calibration information for both high speed, and high resolution converters.

Additionally, there are distinct sets of positive slope(Pslope), negative slope(Nslope), Center, and Offset values for each of the 4 gain settings on the device.

High speed AIN calibration values **HS[4]**:
HS[0] = calibration for gain x1
HS[1] = calibration for gain x10
HS[2] = calibration for gain x100
HS[3] = calibration for gain x1000

High resolution (-Pro only) AIN calibration values **HR[4]**:
HR[0] = calibration for gain x1
HR[1] = calibration for gain x10
HR[2] = calibration for gain x100
HR[3] = calibration for gain x1000

21.0 SD Card



The microSD card is only useful for people who are using autonomous scripting, since all other interactions with the T7 can be saved to the host PC hard drive.

As of this writing, the T7-Pro ships with a 2-4GB micro SD card (SLC technology), RTC, and battery/battery-holder installed. The T7 has none of these, but does have the socket installed to hold an SD card. The retainer opens by sliding the metal piece forward, then lifting.

Currently micro SDXC is not supported. Generally speaking, anything above 2GB is 'HC' meaning high capacity, and HC cards might need to be reformated before they work.

The T7 supports FAT and FAT32 file systems, but some makes and sizes behave differently. We recommend the following SD card format:

```
File System: FAT
Allocation unit size: 64 kilobytes
```

FAT32 with an allocation unit size of 16kb or 32kb sometimes works, but smaller allocation sizes generally do not. On 2GB cards it's possible to select FAT format with a 32kb allocation size, and that sometimes works.

Care must be taken to ensure that power is not lost during file writing or disk corruption could occur. The rated operating temperature of the SD card is -25°C to 85°C. For extremely low temperatures, customers can buy industrial grade SD cards, such as the AF1GUDI-OEM, from ATP Electronics, Inc.

File and directory names are limited to ASCII characters only. Requires firmware 1.0134 or greater. One way to download files from the SD card is to remove it from the device. We also have a beta program for downloading files from the SD card. Note that the beta program is subject to changes, and will not work with firmware older than 1.0150. Using the beta file downloader program linked above, file transfer speeds are 41kB/s via USB, and about 80kB/s via Ethernet. So a 2.5MB file will take 1 minute to download via USB, and 30s via Ethernet.

Get the name of the current working directory (CWD)

1. Write a value of 1 to FILE_IO_DIR_CURRENT. The error returned indicates whether there is a directory loaded as current. No error (0) indicates a valid directory.
2. Read FILE_IO_NAME_READ_LEN.
3. Read an array of size FILE_IO_NAME_READ_LEN from FILE_IO_NAME_READ.
4. Resultant string will be something like "/" for the root directory, or "/DIR1/DIR2" for a directory.

Get list of items in the CWD

1. Write a value of 1 to FILE_IO_DIR_FIRST. The error returned indicates whether anything was found. No error (0) indicates that something was found. FILE_IO_NOT_FOUND (2960) indicates that nothing was found.
2. Read FILE_IO_NAME_READ_LEN, FILE_IO_ATTRIBUTES, and FILE_IO_SIZE. Store the attributes and size associated with each file.

3. Read an array from FILE_IO_NAME_READ of size FILE_IO_NAME_READ_LEN. This is the name of the file/folder.
4. Write a value of 1 to FILE_IO_DIR_NEXT. The error returned indicates whether anything was found. No error (0) indicates that there are more items->go back to step 2. FILE_IO_NOT_FOUND (2960) indicates that there are no more items->Done.

Change the CWD

1. Find from the list of items a directory to open, e.g. "/DIR1". Directories can be parsed out of the list of items by analyzing their FILE_IO_ATTRIBUTES bitmask. If bit 1 of the FILE_IO_ATTRIBUTES bitmask is set, then the item is a directory.
2. Write the directory name length in bytes to FILE_IO_NAME_WRITE_LEN (ASCII, so each char is 1 byte, also don't forget to add 1 for the null terminator).
3. Write the directory string (converted to an array of bytes, with null terminator) to FILE_IO_NAME_WRITE. (array size = length from step 2)
4. Write a value of 1 to FILE_IO_DIR_CHANGE.
5. Done. Optionally get a list of items in the new CWD.

Get disk size and free space

1. Read FILE_IO_DISK_SECTOR_SIZE, FILE_IO_DISK_SECTORS_PER_CLUSTER, FILE_IO_DISK_TOTAL_CLUSTERS, FILE_IO_DISK_FREE_CLUSTERS. Disk operations are performed when you read sector size, all others are snapshots.
2. Total size = sector_size * sectorsPerCluser * Total_Clusters. Free size = sector_size * sectorsPerCluser * free_Clusters

Read a file

1. Write the length of the file name to FILE_IO_NAME_WRITE_LEN (add 1 for the null terminator)
2. Write the name to FILE_IO_NAME_WRITE (with null terminator)
3. Read from FILE_IO_OPEN
4. Read file data from FILE_IO_READ (using the size from FILE_IO_SIZE)
5. Write a value of 1 to FILE_IO_CLOSE

Pass file or directory names to the navigation registers to move around in the file structure.

FILE IO Navigation

Name	Start Address	Type	Access	Default
FILE_IO_NAME_WRITE_LEN	60640	UINT32	W	
FILE_IO_NAME_READ_LEN	60642	UINT32	R	
FILE_IO_NAME_WRITE	60650	BYTE	W	
FILE_IO_NAME_READ	60652	BYTE	R	

FILE_IO_NAME_WRITE_LEN

FILE_IO_NAME_READ_LEN

FILE_IO_NAME_WRITE

FILE_IO_NAME_READ

FILE IO File Operations

Name	Start Address	Type	Access	Default
FILE_IO_OPEN	60620	UINT16	W	
FILE_IO_CLOSE	60621	UINT16	W	
FILE_IO_WRITE	60654	BYTE	W	
FILE_IO_READ	60656	BYTE	R	
FILE_IO_DELETE	60622	UINT16	W	
FILE_IO_ATTRIBUTES	60623	UINT16	R	
FILE_IO_SIZE	60628	UINT32	R	

FILE_IO_OPEN

FILE_IO_CLOSE

FILE_IO_WRITE

FILE_IO_READ

FILE_IO_DELETE

FILE_IO_ATTRIBUTES

FILE_IO_SIZE

FILE IO Directory Operations

Name	Start Address	Type	Access	Default
FILE_IO_DIR_FIRST	60610	UINT16	W	
FILE_IO_DIR_NEXT	60611	UINT16	W	
FILE_IO_DIR_CHANGE	60600	UINT16	W	
FILE_IO_DIR_CURRENT	60601	UINT16	W	
FILE_IO_DIR_MAKE	60602	UINT16	W	
FILE_IO_DIR_REMOVE	60603	UINT16	W	

FILE_IO_DIR_FIRST

FILE_IO_DIR_NEXT

FILE_IO_DIR_CHANGE

FILE_IO_DIR_CURRENT

FILE_IO_DIR_MAKE

FILE_IO_DIR_REMOVE

Read from the disk information registers to get free space and other information.

FILE IO Disk Information

Name	Start Address	Type	Access	Default
FILE_IO_DISK_SECTOR_SIZE	60630	UINT32	R	
FILE_IO_DISK_SECTORS_PER_CLUSTER	60632	UINT32	R	
FILE_IO_DISK_TOTAL_CLUSTERS	60634	UINT32	R	
FILE_IO_DISK_FREE_CLUSTERS	60636	UINT32	R	
FILE_IO_DISK_FORMAT	60638	UINT32	R	

FILE_IO_DISK_SECTOR_SIZE

FILE_IO_DISK_SECTORS_PER_CLUSTER

FILE_IO_DISK_TOTAL_CLUSTERS

FILE_IO_DISK_FREE_CLUSTERS

FILE_IO_DISK_FORMAT

The name operations are where strings/arrays are handled. The registers that are type BYTE are AAI binary.

SD Card Access in Lua Scripts

While running a Lua script and also using a WiFi connection, files on the SD card should not be accessed any faster than once every 5 seconds. Accessing SD card files from within a Lua script at fast rates may prevent WiFi reconnect attempts when the WiFi signal is lost or interrupted. Ethernet and USB unaffected.

Use the following Lua psuedocode to write to a file once every 5 seconds, and read an analog input once every 500ms. See the "Log voltage to file" Lua examples in Kipling for an actual script example.

```
LJ.IntervalConfig(0, 500)  --set the DAQ interval to 500ms, should divide evenly into file access interval
LJ.IntervalConfig(1, 5000)  --set the file access interval to 5 seconds

TableSize = 5000/500
data = {}
DAQcount = 0

for i=1, TableSize do
  data[i] = 0
end

while true do
  if LJ.CheckInterval(0) then --if a data point needs to be collected
    data[DAQcount] = MB.R(0, 3)--collect a new reading from AIN0
    DAQcount = DAQcount + 1
  end
  if LJ.CheckInterval(1) then --file access interval complete
    appendToFile(data)          --save the data  to a file
    DAQcount = 0
  end
end
```

22.0 OEM Versions

For pricing/ordering, go to the main T7 Product Page.

The OEM version of the T7 and T7-Pro are shown below. The enclosure, and most connectors are not installed on the OEM version, which allows customers to choose custom connectors.

The following list describes parts that we know to be compatible with the T7 OEM hole patterns. Simply select a connector from each category, and we can order the parts and construct a custom OEM. Custom OEM boards carry additional cost, but they are often necessary for specialized enclosures, and seamless integration with other products.

Of course there are many other connector options available; we can just as easily order/install something not mentioned below. Please don't hesitate to contact us.

The PCB Dimensions can be found in the Enclosure and PCB Drawings section.

Note: Proper ESD precautions should be taken when handling the PCB directly. Many of the parts are ESD resistant, but depending on the size of the shock, or location, the board might be damaged.

USB

The USB connector is not installed on the T7 OEM. Reference the T7 PCB dimensions for mechanical mating details. Many through-hole Type-B USB connectors are compatible. On Shore Technology Inc USB-B1HSW6, FCI 61729-0010BLF, and TE Connectivity 292304-2 are all good options. The USB connector must be installed on the component side of the PCB.

A special high-retention connector such as the Samtec USBR-B-S-S-O-TH can also be used, but it does take a good deal of force to unplug a cable from these so they are only recommended when you don't want to unplug very often.

It is also possible to simply solder the wires directly, using the image below as a reference.

A normal USB cable has a shield, and the normal Type-B connector connects the cable shield to the mounting tabs on the connector which are then soldered into the large USB mounting holes on the PCB. If you are not using a normal USB connector, and have a shield in your USB cable, we recommend that the shield be connected to either of the large USB mounting holes on the PCB. Usually the USB shield wires are aluminum, which doesn't take solder very well, so use a crimp connector like the Molex 02-06-2103, TE 61388-1, TE 350015-2, or the TE 60017-3. Secure the crimp connector to USB shield wires, then squish down the tip of the connector to fit into the large USB mounting holes on the PCB.

J5 - Alternate Power Supply

The USB connector is a good way to bring in power, even with some sort of custom cable, but if you do not want the USB connector installed J5 provides alternate connections to supply 5V to the T7. The square shaped pad is V+, and the circular pad is GND. It is useful for individuals who only need Ethernet or WiFi. The J5 connector is a 2 pin 0.1" pitch rectangular header. To prevent accidentally

switching V+ and GND, use a keyed connector such as TE Connectivity 3-641215-2.

	J5					
	1	V+		2	GND	
	J5 OEM Pin-Header					

The 5V supply from J5 goes through R21 (0.1 ohms) and then connects to the device-wide VS bus. The 5V supply from USB goes through R15 (0.1 ohms) and then connects to VS. On the T7-T7Pro, **R15 & R21 are both installed by default, and thus the connections for both sources are essentially shorted to each other,** and both should not be connected at the same time as one could back-feed the other. If you are going to connect to J5, and there is a possibility of power at the USB connection also, remove R15. You can also replace R15 and R21 with diodes (SMA package) to prevent back-feeding, but even Schottky diodes will have voltage drop that needs to be considered.

Ethernet

The same Ethernet connector is installed on all versions of the T7 due to the inherent magnetic complexities. However, it is possible to 'bring out' a duplicate Ethernet jack to any custom enclosure with one of the following:

- A short Ethernet cable segment and an RJ45 coupler(Plug to Plug). These couplers come in a few varieties: Free hanging (in-line), Chassis Mount, Panel Mount, Bulkhead, Wall Plate, etc. Conec 33TS3101S-88N and Emerson 30-1008KUL are both good options.
- A RJ45 Jack to Plug cable, which is just a standard Ethernet plug on one end, and a Jack (female) on the other end. Again, these come in a wide variety of mounting styles, the simplest of which is the panel mount. TE Connectivity 1546414-4 and Amphenol RJFEZ2203100BTX are both good options.

If selecting your own Ethernet interconnect, insure that it is RJ45, straight-through, and without magnetics.

WiFi Antenna

T7-Pro-OEM ships with a simple 30mm U.FL whip antenna such as the Anaren 66089-2406. See "Antenna Details" in the WiFi section of this datasheet for additional information.

JP1-JP6 - Screw terminal Locations

The screw terminals are not installed on the OEM T7. Customers will typically use the rectangular header locations (J2, J3) instead of the screw terminals. However, if a different screw terminal style is required, it is possible to buy an OEM T7 and order a custom variety. The screw terminal holes are compatible with almost all 4 position, 0.198" (5.00mm) pitch terminal blocks. A Weidmuller 9993300000 works quite well, and accepts 14-24 AWG wire.

P2, P3 - DB(D-Sub) Locations

The DB15 and DB37 connectors are not installed on an OEM T7. Customers will typically use the rectangular header locations (J2, J3) instead of the DB connectors. However, if a different DB mating style is required, it is possible to buy an OEM T7 and order a custom variety. The DB connectors are standard D-Sub two row receptacles(female sockets), through hole, 15 pin, and 37 pin. The following represent a few valid options.

- FCI 10090099-S154VLF
- FCI D15S33E4GV00LF
- Sullins Connector Solutions SDS101-PRW2-F15-SN13-1
- FCI 10090099-S374VLF
- FCI D37S33E4GV00LF
- Sullins Connector Solutions SDS101-PRW2-F37-SN83-6

J2, J3 - Header Locations

Connectors J2 and J3 provide pin-header alternatives to the DB15 and DB37 connectors. The J2 and J3 holes are always present, but are obstructed when the DB15 and DB37 are installed.

J2 - 16 position, 2 row, 0.1" pitch, male pin rectangular header

- Unshrouded - Harwin Inc M20-9980846
- Unshrouded 3x Taller - Samtec Inc TSW-108-17-T-D
- Shrouded, Gold Finish - On Shore Technology Inc 302-S161
- Shrouded, Right Angle - TE Connectivity 1-1634689-6

J3 - 40 position, 2 row, 0.1" pitch, male pin rectangular header

- Unshrouded - Harwin Inc M20-9762046
- Unshrouded 3x Taller - Samtec Inc TSW-120-17-T-D
- Shrouded, Gold Finish - On Shore Technology Inc 302-S401
- Shrouded, Right Angle - TE Connectivity 5103310-8
- Shrouded, Gold-Palladium Finish - TE Connectivity 5104338-8

Sometimes customers order tall pin headers that mate directly to a separate custom PCB. Refer to the pinout details below for electrical connections.

	J2							
	1	GND		2	VS			
	3	CIO0		4	CIO1			
	5	CIO2		6	CIO3			
	7	GND		8	EIO0			
	9	EIO1		10	EIO2			
	11	EIO3		12	EIO4			
	13	EIO5		14	EIO6			
	15	EIO7		16	GND			
	J2 OEM Pin-Header							
	J3							
	1	GND		2	GND		3	PIN20 (10uA)
	4	PIN2 (200uA)		5	FIO7		6	FIO6
	7	FIO5		8	FIO4		9	FIO3
	10	FIO2		11	FIO1		12	FIO0
	13	MIO0/CIO0		14	MIO1/CIO1		15	MIO2/CIO2
	16	GND		17	Vs		18	Vm-
	19	Vm+		20	GND		21	DAC1
	22	DAC0		23	GND		24	AIN13
	25	AIN12		26	AIN11		27	AIN10
	28	AIN9		29	AIN8		30	AIN7
	31	AIN6		32	AIN5		33	AIN4
	34	AIN3		35	AIN2		36	AIN1
	37	AIN0		38	GND		39	GND
	40	GND						
	J3 OEM Pin-Header							

J4 - Constant Current Sources

Since the screw terminals are not installed on an OEM T7, the J4 header location can be used to gain access to the constant current sources. Any 6 position 0.1" pitch rectangular header will work.

	J4				
	1	200uA		2	GND
	3	GND		4	GND
	5	10uA		6	VS
	J4 OEM Pin-Header				

J8 - Mechanical

The J8 pin header location is purely for mechanical support for that region of the board. There are no electrical connections to this area. It is a 2 position 0.1" pitch rectangular header.

Pricing/Ordering

For pricing & ordering, go to the main T7 Product Page.

23.0 Watchdog

The Watchdog system can perform various actions if the T7 does not receive any communication within a specified timeout period.

A typical usage is to first use the IO Config system to set the power-up defaults for everything as desired, and then configure the watchdog to reset the device on timeout. For example, consider a software program that enables the watchdog to reset the T7 with a 60 second timeout, and then the software has a loop that talks to the device once per second. If something goes wrong with the software, or some other problem that causes communication to stop, the T7 will reset every 60 seconds until communication resumes.

The watchdog timeout can be set as low as 1 second, but such a low value is usually not a good idea. For example, when a USB device resets it takes a little time for USB to re-enumerate and software to be able to talk to the device again, so you could get in a situation where the device keeps resetting so often that you can't start talking to it again. This might require using the reset-to-factory jumper (FIO2 <=> SPC).

Watchdog Registers

Name	Start Address	Type	Access	Default
WATCHDOG_ENABLE_DEFAULT	61600	UINT32	R/W	0
WATCHDOG_ADVANCED_DEFAULT	61602	UINT32	R/W	0
WATCHDOG_TIMEOUT_S_DEFAULT	61604	UINT32	R/W	0
WATCHDOG_STARTUP_DELAY_S_DEFAULT	61606	UINT32	R/W	0
WATCHDOG_STRICT_ENABLE_DEFAULT	61610	UINT32	R/W	0
WATCHDOG_STRICT_KEY_DEFAULT	61612	UINT32	R/W	0
WATCHDOG_STRICT_CLEAR	61614	UINT32	W	0
WATCHDOG_RESET_ENABLE_DEFAULT	61620	UINT32	R/W	0
WATCHDOG_DIO_ENABLE_DEFAULT	61630	UINT32	R/W	0
WATCHDOG_DIO_STATE_DEFAULT	61632	UINT32	R/W	0
WATCHDOG_DIO_DIRECTION_DEFAULT	61634	UINT32	R/W	
WATCHDOG_DIO_INHIBIT_DEFAULT	61636	UINT32	R/W	0
WATCHDOG_DAC0_ENABLE_DEFAULT	61640	UINT32	R/W	0
WATCHDOG_DAC0_DEFAULT	61642	FLOAT32	R/W	0
WATCHDOG_DAC1_ENABLE_DEFAULT	61650	UINT32	R/W	0
WATCHDOG_DAC1_DEFAULT	61652	FLOAT32	R/W	0

WATCHDOG_ENABLE_DEFAULT
Write a 1 to enable the watchdog or a 0 to disable. The watchdog must be disabled before writing any of the other watchdog registers (except for WATCHDOG_STRICT_CLEAR).

WATCHDOG_ADVANCED_DEFAULT
A single binary-encoded value where each bit is an advanced option. If bit 0 is set, IO_CONFIG_SET_CURRENT_TO_FACTORY will be done on timeout. If bit 1 is set, IO_CONFIG_SET_CURRENT_TO_DEFAULT will be done on timeout.

WATCHDOG_TIMEOUT_S_DEFAULT
When the device receives any communication over USB/Ethernet/WiFi, the watchdog timer is cleared. If the watchdog timer is not cleared within the timeout period, the enabled actions will be done.

WATCHDOG_STARTUP_DELAY_S_DEFAULT
This specifies the initial timeout period at device bootup. This is used until the first time the watchdog is cleared or timeout ... after that the normal timeout is used.

WATCHDOG_STRICT_ENABLE_DEFAULT
Set to 1 to enable strict mode.

WATCHDOG_STRICT_KEY_DEFAULT
When set to strict mode, this is the value that must be written to the clear register.

WATCHDOG_STRICT_CLEAR
When running in strict mode, writing the key to this register is the only way to clear the watchdog.

WATCHDOG_RESET_ENABLE_DEFAULT
Timeout action: Set to 1 to enable device-reset on watchdog timeout.

WATCHDOG_DIO_ENABLE_DEFAULT
Timeout action: Set to 1 to enable DIO update on watchdog timeout.

WATCHDOG_DIO_STATE_DEFAULT
The state high/low of the digital I/O after a Watchdog timeout. See DIO_STATE

WATCHDOG_DIO_DIRECTION_DEFAULT
The direction input/output of the digital I/O after a Watchdog timeout. See DIO_DIRECTION

WATCHDOG_DIO_INHIBIT_DEFAULT
The inhibit mask of the digital I/O after a Watchdog timeout. See DIO_INHIBIT

WATCHDOG_DAC0_ENABLE_DEFAULT
Timeout action: Set to 1 to enable DAC0 update on watchdog timeout.

WATCHDOG_DAC0_DEFAULT
The voltage of DAC0 after a Watchdog timeout.

WATCHDOG_DAC1_ENABLE_DEFAULT
Timeout action: Set to 1 to enable DAC1 update on watchdog timeout.

WATCHDOG_DAC1_DEFAULT
The voltage of DAC1 after a Watchdog timeout.

Example

The most common way to use Watchdog is to write:

WATCHDOG_ENABLE_DEFAULT=0
WATCHDOG_TIMEOUT_S_DEFAULT=60
WATCHDOG_RESET_ENABLE_DEFAULT=1
WATCHDOG_ENABLE_DEFAULT=1

If the device does not receive any communication for 60 seconds, the watchdog will cause the device to reset. So if nothing is talking to the device, it will reset every 60 seconds. In conjunction you would often use the IO Config system to configure the power-up defaults as desired.

24.0 IO Config, _DEFAULT

_DEFAULT: Any register with _DEFAULT at the end is non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle.

IO CONFIG: IO Config is a system that concerns the configuration of many registers, mostly related to I/O on the device. This system includes all writable registers for AIN, DAC, and DIO, among others. IO Config does not include registers that have a _DEFAULT version, which is ETHERNET, WIFI, and WATCHDOG, among others.

- **Default:** Values at reboot/power-up.
- **Current:** Current values.
- **Factory:** Factory values.

IO Config Registers

Name	Start Address	Type	Access	Default
IO_CONFIG_SET_DEFAULT_TO_CURRENT	49002	UINT32	W	
IO_CONFIG_SET_DEFAULT_TO_FACTORY	49004	UINT32	W	
IO_CONFIG_SET_CURRENT_TO_FACTORY	61990	UINT16	W	
IO_CONFIG_SET_CURRENT_TO_DEFAULT	61991	UINT16	W	

IO_CONFIG_SET_DEFAULT_TO_CURRENT
Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Current values are retrieved and saved as the new defaults.

IO_CONFIG_SET_DEFAULT_TO_FACTORY
Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Factory values are retrieved and saved as the new defaults.

IO_CONFIG_SET_CURRENT_TO_FACTORY
Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers.

IO_CONFIG_SET_CURRENT_TO_DEFAULT
Write a 1 to set current values to default configuration. The default values are retrieved from flash and written to the current configuration registers, thus this behaves similar to reboot/power-up.

Example

Use normal current configuration registers to write some values, and then save those as defaults so they are in effect at power-up:

AIN_ALL_RANGE = 0.1 //Set current range of all AIN to +/-0.1V
AIN_ALL_RESOLUTION_INDEX = 12 //Set current resolution index of all AIN to 12.
IO_CONFIG_SET_DEFAULT_TO_CURRENT = 1 //Set power-up defaults to current values.

25.0 Scripting

Scripting is relatively new(Nov, 2014), please update your firmware to at least 1.0134

Additions/Changes as of firmware v1.0134:

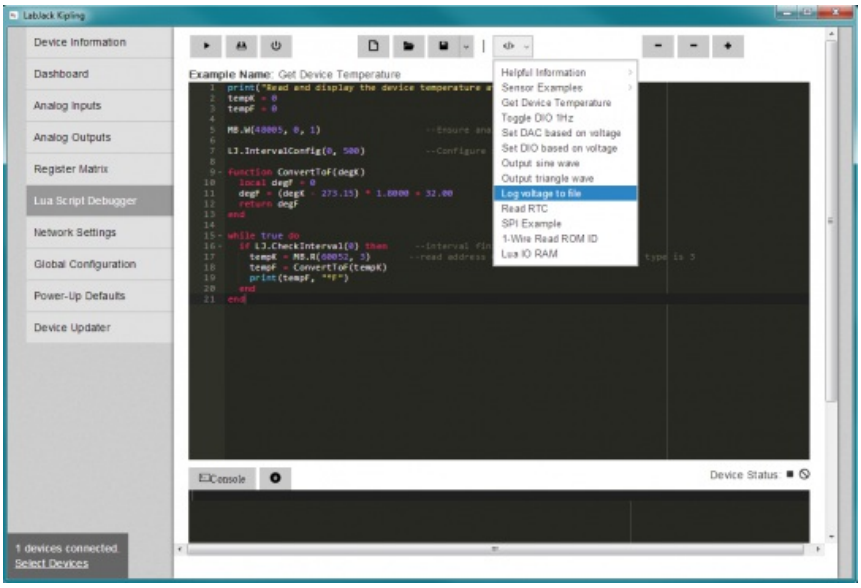
- In firmware v1.0134 and newer, syntax errors and undefined variables no longer reset the T7. It is highly recommended to use FW 1.0134 or newer for scripting.
- Starting with FW 1.0134, the File IO system can create files on the SD card (instead of just using existing files)
- Starting with FW 1.0128, reading the RTC returns the correct time
- Added the ability to run a script when the T7 starts up.
- New firmware solves a bug with converting constants during compilation.
- Starting with FW 1.0059, a jumper between SPC and FIO2 will turn off the script at startup. This hardware-level restore defaults feature is very helpful if the currently loaded Lua script is constantly resetting the device.

The T7 can execute Lua code to allow independent operation. This can be used to collect data without a host computer or perform complex tasks producing simple results that a host can read. And probably other things we haven't thought of.

Getting Started:

1. Get, or launch Kipling v3.

2. Make sure your T7 has firmware v1.0100 or later. Minimum of 1.0134 is recommended.

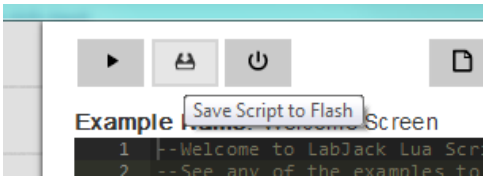


3. Launch Kipling v3, and navigate to the Lua Script Debugger. Connect your T7 to your computer and press run. Open the Get Device Temperature example, and click Run. Now click Stop. Note that clicking stop clears the Lua VM, so even if a program has concluded, it is still necessary to press the stop button.

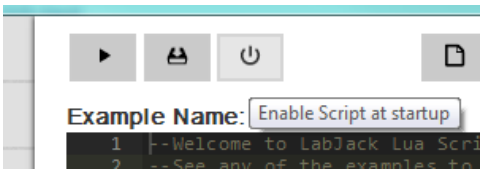
4. Try out some other examples.

Running a script when the T7 powers up:

The T7 can be configured to run a script when it powers on or resets. Typically scripts are tested for a while with the Run/Stop button, while viewing the debug output in the console. Then once everything is working correctly, users will enable the script at startup and close Kipling.



1. Click Save Script to Flash.



2. Click Enable Script at Startup. Now when the T7 is powered on or reset, it will run your script.

Learning more about Lua:

Learning Lua is very easy. There are good tutorials on Lua.org as well as several other independent sites. If you are familiar with the basics of programming such as loops and functions then you should be able to get going just by looking at the examples. If you have suggestions or comments, please email support@labjack.com.

Not sure how to accomplish a goal:

Shoot us an email. We will make a new example and add functions as necessary.

Some things to keep in mind while writing Lua for the T7:

- Try to keep names short. String length directly affects execution speed and code size.

- Based on eLua 0.8 which is based on Lua 5.1.4
- Lua supports muti-return: D, T = LJ.TickDelta(LT). Both D and T are returned values.
- On the T7 Lua's one and only numeric data type is IEEE 754 single precision, aka float. This is more important than it sounds. Here a good article on floating point numbers and their pitfalls: Floating Point Numbers
- Examples contain comments, and currently comments consume a lot of code space. You may want to limit comments if you are making a long script (>200 lines)

LabJack's Lua library:

Data Types - Same types as the LJM Library

- **0** - unsigned 16-bit integer
- **1** - unsigned 32-bit integer
- **2** - signed 32-bit integer
- **3** - single precision floating point (float)
- **98** - string
- **99** - byte - The "byte" dataType is used to pass arrays of bytes in what Lua calls tables

Available Functions - The basic Lua libraries are extended by a LabJack specific library. Below are the available functions.

MB.R

Value = MB.R(Address, dataType)

Modbus read. Will read a single value from a modbus register. That item can be a u16, u32, a float or a string.

MB.W

MB.W(Address, dataType, value)

Modbus write. Writes a single value to a modbus register. The type can be a u16, u32, a float, or a string.

MB.WA

error = MB.WA(Address, dataType, nValues, table)

Modbus write array. Reads nValues from the supplied table, interprets them according to the dataType and writes them as an array to the register specified by Address. The table must be indexed with numbers from 1 to nValues.

MB.RA

table, error = MB.RA(Address, dataType, nValues)

Modbus read array. Reads nValues of type dataType from Address and returns the results in a Lua table. The table is indexed from 1 to nValues.

LJ.ledtog

Toggles status LED. This is just for testing and will be removed.

LJ.Tick

Ticks = LJ.Tick()

// Reads the core timer. (1/2 core freq).

LJ.IntervalConfig & LJ.CheckInterval

IntervalConfig and CheckInterval work together to make an easy to use timing function. Set the desired interval time with IntervalConfig, then use CheckInterval to watch for timeouts. The interval period will have some jitter but no overall error. Jitter is typically ±30 µs but can be greater depending on processor loading. A small amount of error is induced when the processor's core speed is changed.

Up to 8 different intervals can be active at a time.

LJ.IntervalConfig(handle, time_ms)

handle: 0-7

time_ms: Number of milliseconds per interval.

timeout LJ.CheckInterval(handle)

handle: 0-7

Returns: 1 if the interval has expired. 0 if not.

Example:

```
LJ.IntervalConfig(0, 1000)
while true do
  if LJ.CheckInterval(0) then
    --Code to run once per second here.
  end
end
```

LJ.setLuaThrottle

LJ.setLuaThrottle(newThrottle)

Set the throttle setting. This controls Lua's processor priority. Value is number of Lua instruction to execute before releasing control to the normal polling loop. After the loop completes Lua will be given processor time again.

LJ.getLuaThrottle

ThrottleSetting = LJ.getLuaThrottle()

Reads the current throttle setting

Lua IO Memory

Lua IO Memory consists of a list of modbus addresses where data can be sent to and read from a Lua script.

Data transferring from the Lua script to modbus is handled with simple RAM. Lua writes to the locations and another modbus master can read that information.

Data transferring to the Lua script, from modbus, is handled differently. When a Lua_IO_Write address is written to the address and data are stored in a linked list. The linked list is read as a FIFO by the Lua script. This prevents issues arising from write order and multiple writes to the same address. The dataType currently associated with these operations is Float, F32.

The number of floats dedicated to data transferring to Lua is specified. Note that RAM usage is four times this number.

Lua IO Memory

Name	Start Address	Type	Access	Default
LUA_NUM_IO_FLOATS	6006	UINT32	R/W	

LUA_IO#(0:127)_READ	46000	FLOAT32	R/W
LUA_IO#(0:127)_WRITE	47000	FLOAT32	R/W

LUA_NUM_IO_FLOATS
Allocates memory for x input-output floats.

LUA_IO#(0:127)_READ
User configurable registers for Lua scripts.

Names	Addresses
LUA_IO0_READ, LUA_IO1_READ, LUA_IO2_READ, Show All	46000, 46002, 46004, Show All

LUA_IO#(0:127)_WRITE

Names	Addresses
LUA_IO0_WRITE, LUA_IO1_WRITE, LUA_IO2_WRITE, Show All	47000, 47002, 47004, Show All

$(document).ready(function() { $('.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });

Lua functions:

IOMEM.R

Address, Value = IOMEM.R()

Reads from the FIFO a value written to the 47000 range. Address is zero if FIFO is empty.

IOMEM.W

void IOMEM.W(Address, Value)

Writes to the IO RAM. Values here are simply RAM you can overwrite them and they can be read through modbus at any time.

Example script:

```
MB.W(6006,1,10)
while true do
  add, val = IOMEM.R()
  if add > 0 then
    print(string.format("New MB Write: %0.0f %f", add, val))
    IOMEM.W(add - 1000, val + 100)
  end
end
```

Writing to 6006 sets the number of floats that you would like to allocate to transferring information from Lua. You can now use a host program to write to the 47000 range, the script will display the address and value received then save that value to a read location. The location is the write address - 1000 and 100 is added to the value. Read from the 46000 range to see the data.

Examples in the works: (as of 12/8/2014)

- Low power logging.
- Benchmarking tests.

Future Features:

Firmware Related:

- Save a script to flash
- Read a script from flash
- Write data to web services such as DAQConnect.

Kipling Related:

- Easy toggle between modbus Names and Addresses. Since Addresses are required at the low level, customers must search for what names correspond with what addresses using the modbus map. We hope to integrate this look up process eventually.
- When switching away from the Lua Script Debugger tab, the 'active script' is forgotten. We hope to cache the open script file path, so that users don't have to keep re-opening their script file each time.
- View logged data on µSD card. Currently the easiest way to interact with the file system on the T7s µSD card is to physically remove the SD card from the device by unscrewing the enclosure. Until K3 includes a tool customers can get data off of the µSD card with this beta program. Note that the beta program is subject to changes, and will not work with firmware older than 1.0150.

Lua Known Issues:

- Lua is using a single precision float for its data-type. This means that working with 32-bit integer registers is difficult (see examples below). If any integer exceeds 24-bits the lower bits will be lost. The workaround is to access the modbus register using two numbers, each 16-bits. Lua can specify the data type for the register being written, so if users are expecting a large number that will not fit in a float(>24bits), such as a MAC, then read or write the value as a series of 16-bit integers. If you expect the value to be counting up or down, use MB.RA or MB.RW to access the U32 as a contiguous set of 4 bytes. If the value isn't going to increment, e.g. the MAC address, it is permissible to read it in two separate packets using MB.R.

Read a 32-bit register

```
--Value is expected to be changing, and >24 bits (use MB.RA)
aU32[1] = 0x00
aU32[2] = 0x00
aU32, error = MB.RA(3000, 0, 2)   --DIO0_EF_READ_A. Type is 0 instead of 1
DIO0_EF_READ_A_MSW = aU32[1]
DIO0_EF_READ_A_LSW = aU32[2]

--Value constant, and >16,777,216 (24 bits)
--Read ETHERNET_MAC (address 60020)
MAC_MSW = MB.R(60020, 0)  --Read upper 16 bits. Type is 0 instead of 1
MAC_LSW = MB.R(60021, 0)  --Read lower 16 bits.

--Value <16,777,216 (24 bits)
--Read AIN0_EF_INDEX (address 9000)
AIN0_index = MB.R(9000, 1)  --Type can be 1, since the value will be smaller than 24 bits.
```

Write a 32-bit register

```
--Value might be changed or incremented by the T7, and >24 bits (use MB.WA)
aU32[1] = 0xFF2A
aU32[2] = 0xFB5F
error = MB.WA(44300, 0, 2, aU32) --Write DIO0_EF_VALUE_A. Type is 0 instead of 1

--Value constant, and >24 bits
MB.W(44300, 0, 0xFF2A) --Write upper 16 bits. Type is 0 instead of 1
MB.W(44301, 0, 0xFB5F) --Write lower 16 bits.

--Value <16,777,216 (24 bits)
--Write DIO0_EF_INDEX (address 44100)
MB.W(44100, 1, 7)  --Type can be 1, since the value(7) is smaller than 24 bits.
```

Load Lua Script Manually To Device

Kipling automatically handles loading the script files to the device when users press the Run button. Kipling also automatically reads print statements and other output, and displays them in the console. However, if users want to write their own code to load a script file to the T7, that process is outlined in the pseudocode below. Only attempt this process if you are unhappy with Kipling, because Kipling is cross-platform, and should work for most users.

The LJM functions used in this Pseudocode example are:
- LJM_OpenS
- LJM_eWriteName
- LJM_eWriteNameArray

```
// Configure Lua Script Functionality on T7
```

```
int handle = LJM_OpenS( "ANY", "ANY", "ANY");  // Open the first found device

// Start and Stop a Lua Script:
LJM_eWriteName(handle, "LUA_RUN", 1);  // Start a Lua Script
LJM_eWriteName(handle, "LUA_RUN", 0);  // Stop a Lua Script

// Saving Lua Script to a Device
// 1. Stop the Lua Script
LJM_eWriteName(handle, "LUA_RUN", 0); // Stop the Lua Script
string scriptData = "
LJ.IntervalConfig(0, 1000)
    while true do
    if LJ.CheckInterval(0) then
        print("Hello world, from ",_VERSION,"!\n")
    end
end";
int scriptLength = scriptData.length();
// 2. Write the size of the Lua Script to the device
LJM_eWriteName(handle, "LUA_SOURCE_SIZE", scriptLength);
// 3. Break up the Lua Script array into 32 byte chunks
//    and write them to LUA_SOURCE_WRITE
int numPackets = Math.ceil(scriptLength/32); // Calculate the number of packets to send
int numFullPackets = Math.floor(scriptLength/32); // Calculate the number of full packets to send
int numPartialPackets = numPackets - numFullPackets; // Calculate the number of partial packets to send
for(i = 0; i < numFullPackets; i++) {
    subStr = scriptData.split(i*32, (i+1)*32-1); // get the current packet's data
    dataArray = convertStrToArrayOfDoubles(subStr);
    LJM_eWriteNameArray(handle, "LUA_SOURCE_WRITE", 32, dataArray); // Write data to device
}
// 4. If necessary, write the remaining chunk of data
if(numPartialPackets > 0) {
    firstCharIndex = numFullPackets * 32;
    subStr = scriptData.split(firstCharIndex, scriptLength);
    dataArray = convertStrToArrayOfDoubles(subStr);
    int dataLength = dataArray.length();
    LJM_eWriteNameArray(handle, "LUA_SOURCE_WRITE", dataLength, dataArray); // Write data to device
}
```

Information regarding the functions used in this example are available on LJM's function reference page.

26.0 HTTP Server

HTTP Server (Alpha)

Required firmware: 1.0136.

The T7 has a simple http server that can be used to display data or control the T7.

Web pages are stored on the internal flash chip where there is a maximum of 2MB available to store web pages.

Templating:

Files that end in .htm or .html will be scanned for two tildas surrounding a list of parameters that make a modbus command. When found, the command will be executed and the resulting string will replace everything enclosed in the tildas. The command is a set of parameters separated by commads. Refer to the following list for parameter definitions: Parameter 1: The first parameter is the direction. R for read and W for write. Parameter 2: The second parameter is the modbus address. Parameter 3: The third parameter is the data type. These numbers match the LJM constants in "LabJackM.h". Parameter 4: The fourth parameter tells the T7 how to convert the value to a string.

Example:

SN: ~R,60028,2,1~

Here we are reading from address 60028 (serial number) data type is 32-bit unsigned integer and the string format is also 32-bit unsigned integer. The resulting string is:

SN: 470010531

ReST:

The T7 will respond to ReST commands. The command format is /modbus followed by several parameters., direction (read or write), modbus address, length="number of registers", format=FormatIndex, value=X.XX (for writes only), custom="string"). The custom string is a printf instruction that will be used to change the result string is formatted.

Write Example:

modbus/write/1000/length=2&format=2&value=2.7

Read Example:

modbus/read/0?length=2&format=2&custom=%0.6f

MIMEs:

At the moment only a handful of MIMEs are supported:

- htm
- html
- js
- png
- css

Device Name:

When both the T7 and the computer requesting a web page are on the same network the NBNS can be used instead of an IP. Either the IP address or the device name followed by a '/' can be used to connect to the T7. Examples:

192.168.0.131

my_t7_0531/

Uploading:

To send web pages to the T7 they must be packaged and then loaded into flash. Below are two proof of concept programs. One to package webpages and another to load them onto the T7. The functionality provided by these two utilities will soon be integrated into Kipling.

Know Issues:

- Images currently do not transfer properly.

File attachment:
T7_WebPages_PoC_20151301.zip

Appendix A - Specifications

Specifications for describing the T7 can be broken down into several primary sections with a few sub-sections. Navigate the following sections to see specifications.

A-1 Data Rates

Communication Modes

Communications between the host computer and the T7 occur using one of two modes:

1. Command/Response
2. Stream

The Command/Response mode will apply to most applications and consists of command data packet sent from the host followed by a response data packet from the T7. Stream mode relies on the T7 to carry out periodic sampling events automatically. Collected data is stored in the T7's memory until it retrieved by the host application. Figure A1.1.1 depicts a the two operating modes for the T7.

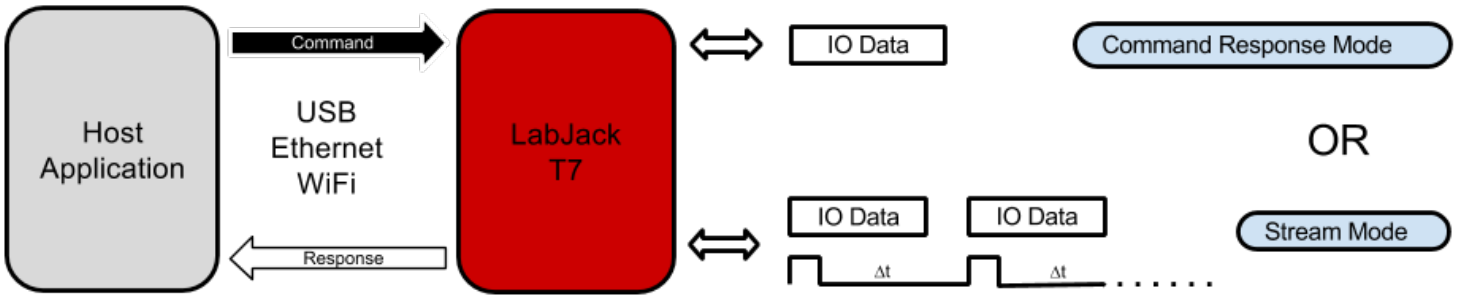


Figure A1.1. T7 communication modes.

The use of a particular mode will depend desired T7 functionality and the hardware response time required by the end application. Not all functionality is supported in stream mode. Please refer to the Stream Mode section of the user's manual, detailing stream mode operations.

Command Response Data Rates

All communication performed with the T7 is accomplished using the Modbus TCP protocol, thus allowing direct communication with the T7 via low-level TCP commands. As an alternative, the LJM library may be used as a higher level communications layer for added convenience and minimal additional overhead. Tables A-1.1 and A-1.2 list expected communication overhead times associated with ModBus TCP and LJM Library communication options, performing various tasks on the T7.

Table A1.1. Typical communication overhead using direct Modbus TCP.

	USB High-High	USB Other	Ethernet	WiFi
	[ms]	[ms]	[ms]	[ms]
No I/O - Overhead	0.7	2.1	0.8	6
Read All DI	0.7	2.1	0.8	6
Write All DO	0.7	2.1	0.8	6
Write Both DACs	0.7	2.1	0.8	6

Table A1.2. Typical communication overhead using LJM library.

	USB High-High	USB Other	Ethernet	WiFi
	[ms]	[ms]	[ms]	[ms]
No I/O - Overhead	1.7	2.2	1.7	7
Read All DI	1.7	2.2	1.7	7
Write All DO	1.7	2.2	1.7	7
Write Both DACs	1.7	2.2	1.7	7

Testing Procedure and Definitions:

The times shown in table A-1.1 and A-1.2 were measured using a LabVIEW program running on Windows were all read and write operations are conducted with a single eNames() call. The eNames() functions is used to minimize the number of Modbus packets sent from the host (one packet per command/response set). The test program executes one of the listed tasks within a loop for a specified number of iterations, over a 1-10 second period. The overall execution time is divided by the total number of iterations, providing the average time per iteration for each task. The execution time includes LabVIEW overhead, LJM library overhead, Windows overhead, communication time (USB/Ethernet/WiFi), and T7 processing time.

A "USB high-high" configuration means the T7 is connected to a high-speed USB2 hub which is then connected to a high-speed USB2 host. Even though the T7 is not a high-speed USB device, such a configuration does provide improved performance. Typical examples of "USB other" would be a T7 connected to an old full-speed hub (hard to find) or more likely the T7 is connected directly to the USB host (your PC) even if the host supports high-speed.

Preemptive Operating Systems and Thread Priority:

It is important to understand that Linux, Mac, and Windows are generally "best-effort" operating systems and not "real-time", meaning that the speeds below can vary based on each individual computer, the hardware inside of it, its currently enabled peripherals, current network traffic, strength of signal, design of the application software, other running software, and many more variables [1].

Ethernet & USB:

These times are quite predictable. Software issues mentioned above are important, but in terms of hardware the times below will be consistent. The T7 is not consuming a major portion of USB or Ethernet bandwidth. Therefore, the overhead times listed are typically maintained even with substantial activity on the bus.

WiFi:

The WiFi times tend to vary much more than USB or Ethernet. With a solid connection most WiFi packets have an overhead of 3-8 ms, but many will take longer. For example, a test was done in a typical office environment of 1000 iterations that produced an average time of 7.0 ms. 92% of the packets took 3-8 ms, 99% took < 30 ms, and 3 packets took 300 ms.
All WiFi tests were done with an RSSI between -40 (very strong) and -70 (good). An RSSI less than -75 generally reflects a weak connection and the number of packets that experiences retries goes up quickly. An RSSI greater than -35 reflects a very strong connection, typically within a few feet of the access point, and also results in increasing numbers of retries due to saturation of the RF signal.

ADC Conversions:

Analog to digital conversions (ADC) will increase the command response time depending on the number of channels, input gain, and resolution index being used. Table A1.3 lists the conversion times for the T7 at various gains and resolution index settings, reading a single analog input channel. The total command response time (CRT) when reading analog inputs is equal to the overhead time from tables A1.1 and A1.2 added to the conversion times from Table A1.3 (per channel being read) . Please review tables A1.1-A1.3 carefully, as the listed times will determine the maximum sampling rate achievable when reading analog inputs in command response mode.

CRT (miliseconds) = overhead + (#AINs * AIN Sample Time)

Table A1.3. Typical milliseconds per sample.

Resolution Index	Effective Resolution	LSB Voltage	AIN Sample Time
	[bits]	[µV]	[ms/sample]
Gain/Range: 1/±10V			
1	16.0	316	0.04
2	16.5	223	0.1
3	17.0	158	0.1
4	17.5	112	0.2
5	17.9	84.6	0.2
6	18.3	64.1	0.3
7	18.8	45.3	0.6
8	19.1	36.8	1.1
9	19.6	26.0	3.5
10	20.5	14.0	13.4
11	21.3	8.02	66.1
12	21.4	7.48	159
Gain/Range:10/±1V			
1	15.4	47.9	0.3
2	16.0	31.6	0.3
3	16.5	22.3	0.6
4	16.9	16.9	0.7
5	17.4	12.0	1.2
6	17.9	8.46	2.3
7	18.3	6.41	2.6
8	18.7	4.86	3.1
9	19.5	2.79	3.5
10	20.5	1.40	13.5
11	21.4	0.748	66.3
12	21.5	0.698	159
Gain/Range: 100/±0.1V			
1	13.3	20.5	1.0
2	14.2	11.0	2.0
3	14.7	7.78	5.0
4	15.2	5.50	5.0
5	15.7	3.89	5.1
6	16.3	2.57	10.3
7	16.7	1.94	10.5
8	17.2	1.37	11.1
9	18.3	0.641	3.5
10	19.1	0.368	13.4
11	19.6	0.260	66.2
12	19.7	0.243	159
Gain/Range: 1000/±0.01V			
1	10.9	10.8	5.0
2	12.3	4.10	10.0
3	12.7	3.11	10.1
4	13.3	2.05	10.1
5	13.8	1.45	10.2
6	14.4	0.96	10.3
7	14.7	0.778	10.6
8	15.0	0.632	11.1
9	15.4	0.479	3.5
10	16.1	0.295	13.4
11	16.4	0.239	66.2
12	16.4	0.239	158

Streaming Data Rates

The fastest data rates on the T7 occur when operating in stream mode. Much of the command response overhead is eliminated in stream mode because the T7 is responsible for initiating IO operations. Collected data is stored in the T7's stream buffer it is retrieved by the host application. The end result is a continuous data stream, sampled at regular intervals, collected with a minimum number of command response data sets [2.]. Table A1.4 and A1.5 provide typical stream-related performance results. The tabulated data is useful for determining what types of signals can be analyzed using a T7. The T7 is capable of streaming analog data at regular discrete intervals. As a result, various discrete time signal analysis tools can be utilized to interpret data.

The scan rates shown in table A1.4 are continuous over USB or Ethernet. When using WiFi, the device can acquire data at the fastest rates, but transfer of data to the host is limited to about 4 ksamples/second, so the fastest stream rates cannot be maintained continuously. In this case stream-burst can be used rather than continuous stream, where each stream is limited to a specified number

of scans that fits in the device's stream buffer.

Table A1.4. Stream scan rates for stream mode over various gain, resolution index, channel count combinations.

	Gain : Range	Maximum Scan Rate			
		1 Channel [Hz]	2 Channels [Hz]	4 Channels [Hz]	8 Channels [Hz]
Resolution Index = 1	1 : ±10V	100k	43.7k	22.5k	11.3k
	10 : ±1V	100k	4.1k	1.4k	585
	100 : ±0.1V	100k	850	315	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 2	1 : ±10V	48k	19.8k	9.0k	4.0k
	10 : ±1V	48k	3.6k	1.3k	550
	100 : ±0.1V	48k	400	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 3	1 : ±10V	22k	9.9k	4.5k	2.4k
	10 : ±1V	22k	1.4k	500	225
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 4	1 : ±10V	11k	4.9k	2.2k	1.3k
	10 : ±1V	11k	1.3k	45	N.S.
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 5	1 : ±10V	5.5k	2.2k	990	630
	10 : ±1V	5.5k	630	23	N.S.
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 6	1 : ±10V	2.5k	1.3k	630	315
	10 : ±1V	2.5k	320	N.S.	N.S.
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 7	1 : ±10V	1.2k	650	315	N.S.
	10 : ±1V	1.2k	220	N.S.	N.S.
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.
Resolution Index = 8	1 : ±10V	600	315	N.S.	N.S.
	10 : ±1V	600	N.S.	N.S.	N.S.
	100 : ±0.1V	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	N.S.	N.S.	N.S.	N.S.

*N.S. indicates settings are not currently supported in stream mode.

A distinction between the terms scan and sample must be drawn to better interpret the data from Table A1.4. A sample and a scan represent two separate parameters which make up a T7 data stream. The definitions for each parameter are as follows:

Sample - A reading from one channel/address.
Scan - One reading from a list of channels/addresses (scan list).

The scan rate, by definition, is a fraction of the sample rate where the fraction is the inverse of the number of channels being read in a single scan. The scan rate is defined as:

ScanRate = SampleRate / NumAddresses

The T7 has a **maximum sample rate** of 100 ksamples/second. The stated maximum sample rate is achievable when a stream is configured with Range = +/-10V and ResolutionIndex = 1. This is reflected in the first row of data in table A1.4 (highlighted). The reported scan rate is simply the maximum sample rate divided by the number of channels in the scan list (within ~10%). Note that the sample rate and scan rate for a single-channel stream are equal since the NumAddresses = 1.

The maximum scan rate decreases at resolution index and range settings other than listed above simply because analog conversions take longer to complete. Table A1.5 illustrates how analog conversion times increase at different resolution index and range settings.

Table A1.5. Stream performance characteristics for single-channel stream over various gain and resolution index combinations.

Resolution Index	Peak-to-Peak Noise [16-bit counts]	Interchannel Delay [µs]
Gain/Range: 1/±10V		
1	±3.0	15
2	±2.0	25
3	±1.5	45
4	±1.0	90
5	±1.0	170
6	±0.5	340
7	±0.5	675
8	±0.5	1,335
Gain/Range: 10/±1V		
1	±4.5	210
2	±3.0	220
3	±2.0	545
4	±1.5	585
5	±1.0	1,200
6	±0.5	2,415
7	±0.5	2,750
8	±0.5	3,415
Gain/Range: 100/±0.1V		
1	±20.0	1,045
2	±14.0	2,105
3	N.S.	N.S.
4	N.S.	N.S.
5	N.S.	N.S.
6	N.S.	N.S.
7	N.S.	N.S.
8	N.S.	N.S.
Gain/Range: 1000/±0.01V		
1	N.S.	N.S.
2	N.S.	N.S.
3	N.S.	N.S.
4	N.S.	N.S.
5	N.S.	N.S.
6	N.S.	N.S.
7	N.S.	N.S.
8	N.S.	N.S.

*N.S. indicates settings are not currently supported in stream mode.

Notes:

1. Various software issues need consideration when implementing a feedback loop that executes at the desired time interval. Some considerations are: thread priority, logging to file, updating the screen, and other programs running on the machine.

2. The number of command response data sets used to retrieve stream data from the T7 depends on the number of data points allowed to accumulate in the in the T7's stream buffer.

A-2 Digital I/O

General Info

Below you can find information regarding the T7's Digital Input/Output lines.

Parameter	Conditions	Min	Typical	Max	Units
Low Level Input Voltage		-0.3		0.5	Volts
High Level Input Voltage		2.64		5.8	Volts
MaximumInput Voltage (1)	FIO	-10		10	Volts
	EIO/CIO/MIO	-6		6	Volts
Output Low Voltage (2)	No Load		0.01		Volts
---FIO	Sinking 1 mA		0.55		Volts
---EIO/CIO	Sinking 1 mA		0.15		Volts
---EIO/CIO	Sinking 5 mA		0.75		Volts
Output High Voltage (2)	No Load		3.3		Volts
---FIO	Sourcing 1 mA		2.75		Volts
---EIO/CIO	Sourcing 1 mA		3.15		Volts
---EIO/CIO	Sourcing 5 mA		2.6		Volts
Short Circuit Current (2)	FIO		6.3		mA
	EIO/CIO/MIO		22.9		mA
Output Impedance (2)	FIO		550		Ω
	EIO/CIO/MIO		180		Ω
(1) Maximum voltage to avoid damage to the device. Protection works whether the device is powered or not, but continuous voltages over 5.8 volts or less than -0.3 volts are not recommened when the T7 is unpowered, as the voltage will attempt to supply operating power to the T7 possibly causing poor start-up behavior.					
(2) These specifications provide the answer to the question. "How much current can the digital I/O sink or source?". For instance, if EIO0 is configured as output-high and shorted to ground, the current sourced by EIO0 is configured as output-high and shorted to ground, the current sourced by EIO0 into ground will be about 16 mA (3.3/180). If connected to a load that draws 5 mA, EIO0 can provide that current but the voltage will droop to about 2.4 volts instead of the nominal 3.3 volts. If connected to a 180 ohm load to ground, the resulting voltage and current will be about 1.65 volts @ 9 mA.					

Extended Features

Below you can find information regarding the T7's DIO EF.

Extended Features	Conditions	Min	Typical	Max	Units
Frequency Output (1)		0.02		5 M	Hz
Counter Input Frequency (2)				5	MHz
Input Timer Total Edge Rate (3)(4)	No Stream			70k	edges/s
	While Streaming @ 50 kHz			20k	edges/s
(1) Frequencies up to 40MHz are possible, but they are heavily filtered.					
(2) Hardware counters. 0 to 3.3 volt square wave.					
(3) To Avoid missing edges, keep the total number of applicable edges on all applicable timers below this limit.					
(4) Excessive processor loading could reduce these limits further.					

Serial Communication

Below you can find information regarding the T7's Serial Communication abilities. Please keep in mind our devices use 3.3V logic levels and provide 5V output along the VS screw terminal. Some ICs require the same logic level as provided to the chip's VCC line so extra steps may be required to integrate specific sensors.

Serial Communication	Conditions	Min	Max	Units
SPI Characteristics				
Clock Frequencies		0.08718	870	kHz
I2C Characteristics				
Clock Frequencies		9.3	472	kHz

A-3 Analog Input

General Info

Below you can find general information regarding the T7's Analog input lines. For further details consult the Noise and Resolution subsection.

	Conditions	Min	Typical	Max	Units
Typical Input Range (1)	Gain=1	-10.5		10.1	Volts
Max AIN Voltage to GND (2)	Valid Readings	-11.5		11.5	Volts
Max AIN Voltage to GND (3)	No Damage	-20		20	Volts
Input Bias Current (4)			20		nA
Input Impedance (4)			1		GΩ
Source Impedance (4)			1		kΩ
Integral Linearity Error	Gain=1, 10, 100			±0.01	%FS
	Gain=1000			±0.1	%FS
Absolute Accuracy	Gain=1, 10, 100			±0.01	%FS
	Gain=1000			±0.1	%FS
Temperature Drift			15		ppm/°C
Noise (Peak-To-Peak)	See A-3-1			<1	µV
Effective Resolution (RMS)	See A-3-1			22	bits
Noise-Free Resolution	See A-3-1			20	bits

(1) Differential or single-ended

(2) This is the maximum voltage on any AIN pin compared to ground for valid measurements on that channel. For single-ended readings on the channel itself, inputs are limited by the "Typical Input Range" above, and for differential readings consult the signal range tables in Appendix-3-2. Further, if a channel has over 13.0 volts compared to ground, readings on other channels could be affected. Because all even channels are on 1 front-end mux, and all odd channels on a 2nd front-end mux, an overvoltage (>13V) on a single channel will generally affect only even or only odd channels.

(3) Maximum voltage, compared to ground, to avoid damage to the device. Protection level is the same whether the device is powered or not.

(4) The key specification here is the maximum source impedance. As long as the source impedance is not over this value, there will be no substantial errors due to impedance problems. For source impedance greater than this value, more settling time might be needed.

A-3-1 Noise And Resolution

A Note About ADC Noise and Resolution

Analog voltages measured by the T7 are converted to digital representation via the T7's analog to digital converter (ADC). The ADC reports an analog voltage in terms of ADC counts, where a single ADC count is the smallest change in voltage that will affect the reported ADC value. A single ADC count is also known as the converter's LSB voltage. The ADC's resolution defines the number of discrete voltages represented over a given input range. For example, a 16-bit ADC with a +/-10 input range can report 65536 discrete voltages (2^{16}) and has an LSB voltage of 0.305 mV (20V ÷ 2^{16}).

The stated resolution for an ADC is a best case value, assuming no channel noise. In reality, every ADC works in conjunction with external circuitry (amplifiers, filters, ect.) which all posses some level of inherent noise. The noise of supporting hardware, in addition to noise of the ADC itself, all contribute to the channel resolution. In general, the resolution for an ADC and supporting hardware will be less than what is stated for the ADC. The combined resolution for an in-system ADC is termed effective resolution. Simply put, the effective resolution is the equivalent resolution where analog voltages less than LSB voltage are no longer differentiable from the inherent hardware noise. In addition to defining the smallest measurable analog voltage, the effective resolution also defines the RMS peak-to-peak noise on a given analog channel.

Closely related to the effective resolution is the error free code resolution (EFCR) or *flicker-free* code resolution. The EFCR represents the resolution on a channel immune to "bounce" or "flicker" from the inherent system noise. The EFCR is not reported in this appendix. However, it may be closely approximated by the following equation:

EFCR = effective resolution - 2.7 bits **[1.]**

The T7 offers user-selectable resolution through the resolution index parameter on any one AIN channel. Internally, the ADC hardware uses modified sampling methods to increase measurement resolution beyond the ADC's base resolution. Valid resolution index values are 0-8 for the T7 and 0-12 for the T7-Pro [2.][3.]. Increasing the resolution index value will improve the channel resolution, but doing so will usually extend channel sampling times. See section 14.0 AIN for more information on the resolution index parameter and its use.

Noise and Resolution Data

The data shown below summarizes typical effective resolutions and expected channel sampling times over all resolution index values. Data for the T7 and T7-Pro data are combined and presented together for convenience, where resolution index values 9-12 only apply to the T7-Pro.

The AIN sampling time is the typical amount of time required for the ADC hardware to make a single analog to digital conversion on any channel and is reported in milliseconds per sample. The AIN sampling time does not include command/response and overhead time associated with the host computer/application.

Noise and Resolution Test procedure

Noise and resolution data was generated by collecting 512 successive voltage readings, using a short jumper between the test channel and ground. The resulting data set represents typical noise measured on any one analog input channel in ADC counts. The effective resolution is calculated by subtracting the RMS channel noise (represented in bits) from 16-bits.

Effective Resolution = 16 bits - $\log_2$ (RMS Noise [in ADC counts])

Table A.3.1.1. Effective resolution and sampling times for various gains and resolution index settings. 9-12=T7-Pro

Resolution Index	Effective Resolution [bits]	Effective Resolution [µV]	AIN Sample Time [ms/sample]
Gain/Range: 1/±10V			
1	16.0	316	0.04
2	16.5	223	0.1
3	17.0	158	0.1
4	17.5	112	0.2
5	17.9	85	0.2
6	18.3	64	0.3
7	18.8	45	0.6
8	19.1	37	1.1
9	19.6	26	3.5
10	20.5	14	13.4
11	21.4	7.5	66.1
12	21.8	5.7	159
Gain/Range:10/±1V			
1	15.4	48	0.3
2	16.0	32	0.3
3	16.5	22	0.6

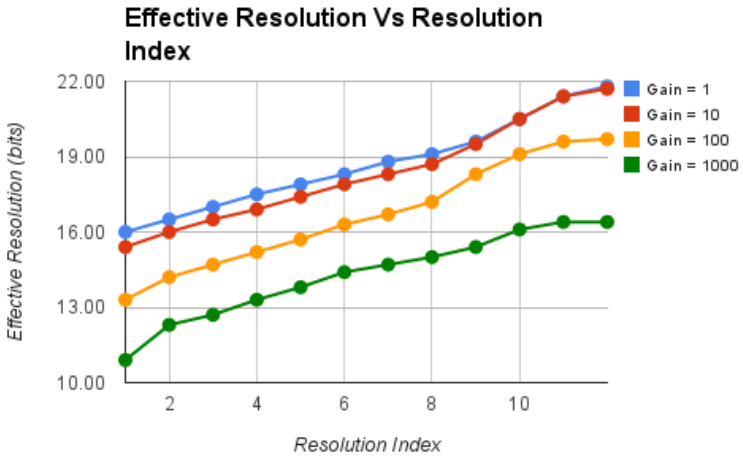


Figure A.3.1.2. Analog input effective resolution over various gains and resolution index settings.

4	16.9	17	0.7
5	17.4	12	1.2
6	17.9	8.5	2.3
7	18.3	6.4	2.6
8	18.7	4.9	3.1
9	19.5	2.8	3.5
10	20.5	1.4	13.5
11	21.4	0.7	66.3
12	21.7	0.6	159
Gain/Range: 100/±0.1V			
1	13.3	21	1.0
2	14.2	11	2.0
3	14.7	7.8	5.0
4	15.2	5.5	5.0
5	15.7	3.9	5.1
6	16.3	2.6	10.3
7	16.7	1.9	10.5
8	17.2	1.4	11.1
9	18.3	0.6	3.5
10	19.1	0.4	13.4
11	19.6	0.3	66.2
12	19.7	0.2	159
Gain/Range: 1000/±0.01V			
1	10.9	11	5.0
2	12.3	4.1	10.0
3	12.7	3.1	10.1
4	13.3	2.1	10.1
5	13.8	1.5	10.2
6	14.4	1.0	10.3
7	14.7	0.8	10.6
8	15.0	0.6	11.1
9	15.4	0.5	3.5
10	16.1	0.3	13.4
11	16.4	0.2	66.2
12	16.4	0.2	158

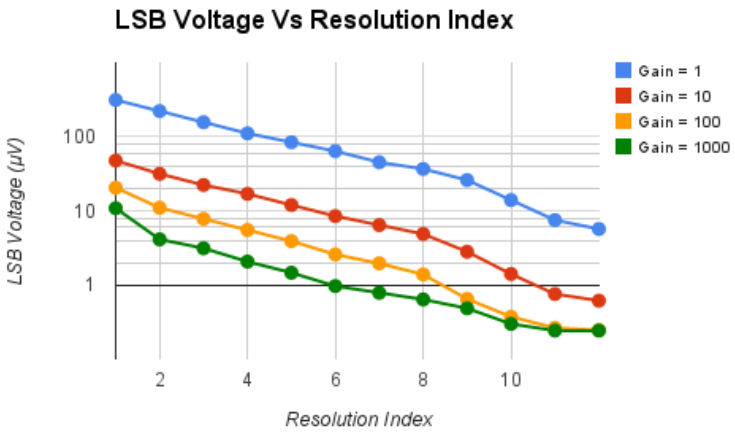


Figure A.3.1.3. Analog input LSB voltage over various gains and resolution index settings.

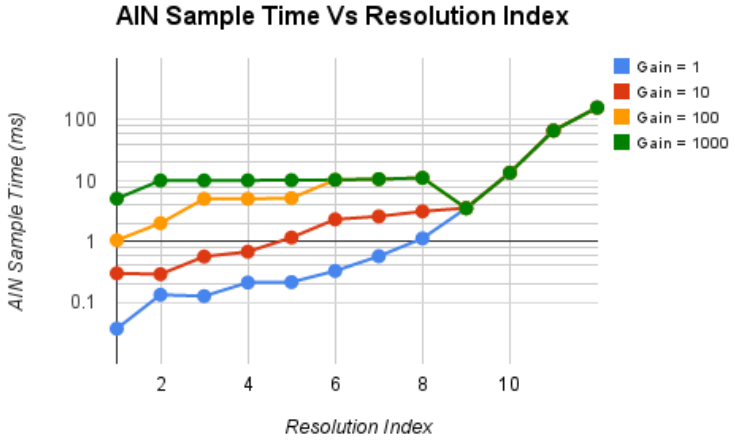


Figure A.3.1.4. AIN sample times for analog inputs over various gains resolution index settings.

Notes:

1. The equation used to approximate the EFCR is determined using +/-3.3 standard deviations from the RMS noise measured on an AIN channel.

2. Resolution index 0 defaults the T7 to resolution index = 8 and the T7-Pro to resolution index = 9.

3. The T7-Pro is equipped with a 24-bit delta-sigma ADC, in addition to the standard 16-bit ADC. Analog conversions occur on the 16-bit ADC when resolution index values 0-8 are used. Analog conversion occur on the 24-bit ADC when resolution index values 9-12 are used.

A-3-2 Signal Range

The following figures show the approximate signal range of the T7 analog inputs. "Input Common-Mode Voltage" or Vcm is (Vpos + Vneg)/2.

Keep in mind that the voltage of any input compared to GND should be within the Vm+ and Vm- rails by at least 1.5 volts, so if Vm is the typical ±13 volts, the signals should be within ±11.5 volts compared to GND.

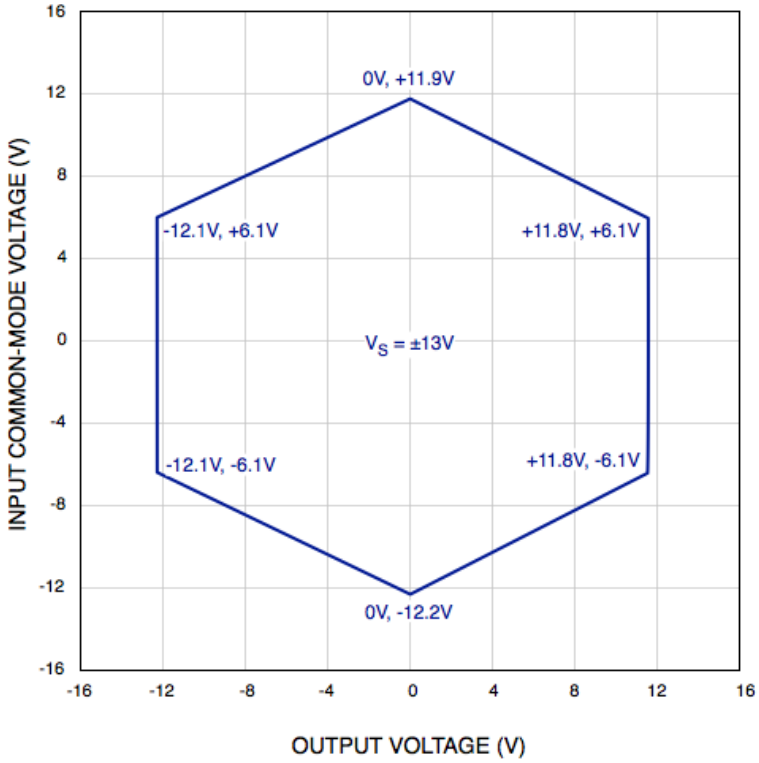
Example #1: Say a differential signal is measured where Vpos is 10.05 volts compared to GND and Vneg is 9.95 volts compared to ground, and G=100. That means Vcm=10.0 volts, Vdiff=0.1 volts, and the expected Vout=10.0 volts. There is not figure for G=100 below, but Vcm=10.0 volts and Vout=10.0 volts is not valid at G=1 or G=1000, so is certainly not valid in between.

Example #2: Say a differential signal is measured where Vpos is 15.0 volts compared to GND and Vneg is 14.0 volts compared to ground, and G=1. That means Vcm=14.5 volts, Vdiff=1.0 volts, and the expected Vout=1.0 volts. The voltage of each input compared to

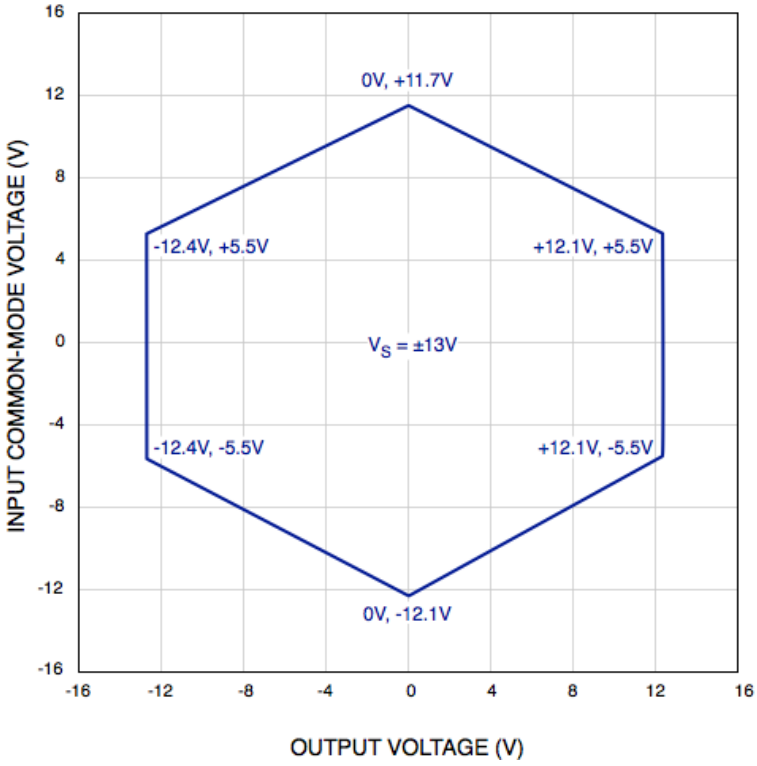
GND is too high, so this would not work at all.

Example #3: Say a single-ended signal is measured where Vpos is 10.0 volts compared to GND and G=1.; That means Vcm=5.0 volts, Vdiff=10.0 volts, and the expected Vout=10.0 volts. This is fine according to the figure below.

Input Common-Mode Voltage Range vs. Output Voltage, G = 1



Input Common-Mode Voltage Range vs. Output Voltage, G = 1000



A-4 Analog Output

The T7 supports two analog output channels labeled "DAC0" and "DAC1". General characteristics of the two channels are available below.

Parameter	Conditions	Min	Typical	Max	Units
Nominal Output Range (1)	No Load	0.01		4.99	Volts
	@ ±2.5 mA	0.25		4.75	Volts
Resolution			12		Bits
Absolute Accuracy	5% to 95% FS		±0.06		% FS
Integral Linearity Error			±1.5	±2	counts
Differential Linearity Error			±0.25	±0.5	counts
Noise (2)			±100		µV
Source Impedance (3)			50		Ω
Current Limit (4)	Max to GND		20		mA
Time Constant			4		µs
(1) Maximum and minimum analog output voltage is limited by the supply voltages (VS and GND). The specifications assume VS is 5.0 volts. Also, the ability of the DAC output buffer to driver voltages close to the power rails, decreases with increasing output current.					
(2) With load, the noise increases if operating too close to VS. With a 1000 ohm load, noise increases noticeably at 4.4V and higher. With a 330 ohm load, noise increases noticeably at 3.7V and higher. With a 100 ohm load, noise increases noticeably at 2.7V and higher.					
(3) For currents up to about 8mA, this source impedance dominates the error due to loading. For example, if you load DAC0 with a 1000 ohm resistor from DAC0 to GND, and set DAC0 to 3.0V, the actual voltage at the DAC0 terminal will be about 3.0*1000/(50+1000) = 2.86V. For currents >8mA, you increasingly get added droop due to the ability of the output buffer to drive substantial current close to the power rails.					
(4) The output buffer will limit current to about 20mA, and can maintain this value continuously without damage. Take for example a 100 ohm resistor from DAC0 to GND, with the internal source impedance of 50 ohms, and DAC0 set to 4.5V. A simple calculation would predict a current of 4.5/(50+150) = 30mA, but the output buffer will limit the current to 20mA. A simple calculation taking into account only the voltage droop due to the internal 50 ohm resistance would predict a voltage at the DAC0 terminal of 4.5*100/(50+100) = 3.0V, but since the current is limited to 20mA the actual voltage at DAC0 would be more like 100*0.02 = 2.0V.					

A-5 General Specs

Supply

The following table shows the supply voltage that is required by the T7. The USB hub, or 5V USB adapter should fall within the acceptable range.

General:Power,Mechanical : Supply

Parameter	*Condition*	*Min*	*Typical*	*Max*	*Units*
Supply Voltage		4.75		5.25	Volts
Supply Current	No connected loads	8.1	250		mA

Power Consumption

The T7 has several power domains. USB and Core speed are not yet ready user level control, but have been included in the following table to show the capabilities of the device. The values shown are typical.

General:Power,Mechanical : Power

Core Speed	Eth (1)	Eth Linked	AINs	WiFi	WiFi Linked	LEDs	USB (1)	Draw (mA)
80M	ON	Yes	ON	ON	Yes	ON	ON	290
80M	ON	Yes	ON	ON	No	ON	ON	285
80M	ON	Yes	ON	OFF	No	ON	ON	253
80M	ON	No	ON	OFF	No	ON	ON	210
80M	OFF	No	ON	OFF	No	ON	ON	174
80M	OFF	No	OFF	OFF	No	ON	ON	142
80M	OFF	No	OFF	OFF	No	OFF	ON	105
80M	OFF	No	OFF	OFF	No	OFF	OFF	79
20M	OFF	No	OFF	OFF	No	OFF	OFF	23
2M	OFF	No	OFF	OFF	No	OFF	OFF	8.8
250k	OFF	No	OFF	OFF	No	OFF	OFF	8.1

1) Ethernet and USB require that the core be running at least 20MHz.

200µA and 100µA Current Sources

General:Power,Mechanical : Current Sources

Parameter	Condition	Min	Typical	Max	Units
Accuracy vs Cal Value (1)	~ 25 °C		±0.1	±0.2	%
Accuracy vs Nominal (1)	~ 25 °C			±5	%
TempCo 200UA (2)	~ 25 °C		-20		ppm/°C
TempCo 10UA (2)	~ 25 °C		-5		ppm/°C
Maximum Voltage			VS - 2.0		volts

(1) First spec is the accuracy compared to the value stored during calibration. The second spec is the accuracy compared to the nominal value (e.g. 200.0 µA for the 200UA source).
(2) Tempco varies strongly with temperature. See the charts in the "200uA and 10uA" datasheet section.

VM+/VM-

General:Power,Mechanical : VM+/VM-

Parameter	Condition	Min	Typical	Max	Units
Typical Voltage	No-load		±13		volts
	@ 2.5 mA		±12		volts
Maximum Current			2.5		mA

System Clock

General:Power,Mechanical : Clock

Parameter	Condition	Min	Typical	Max	Units
Clock Error	~ 25 °C			±20	ppm
	-10 to 60 °C			±50	ppm
	-40 to 85 °C			±100	ppm

Mechanical

General:Power,Mechanical : Mechanical

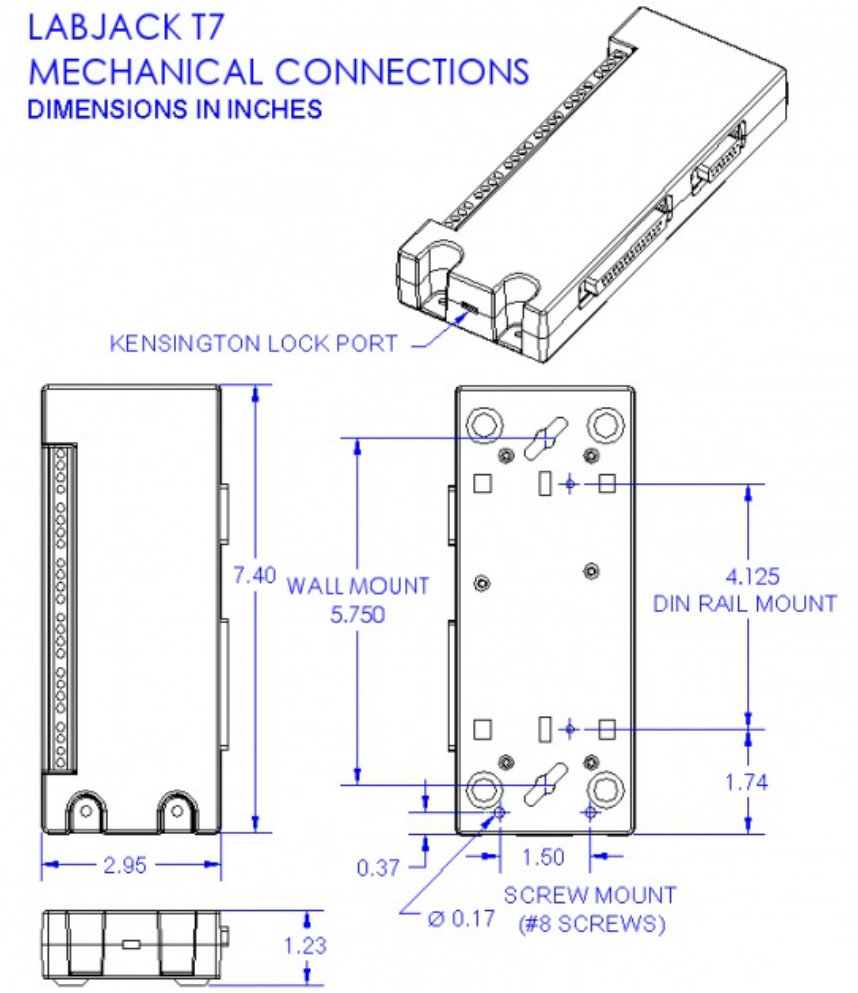
Parameter	Condition	Min	Typical	Max	Units
USB Cable Length			2	5	meters
Operating Temperature		-40		85	°C
Screw Terminal Wire Gauge			26	14	AWG
Mounting Screws	wood screw sizes	#4	#6	#8	
Enclosure Screws (x6)	PH1 pan head		#4-20 x 5/8"		Phil #1

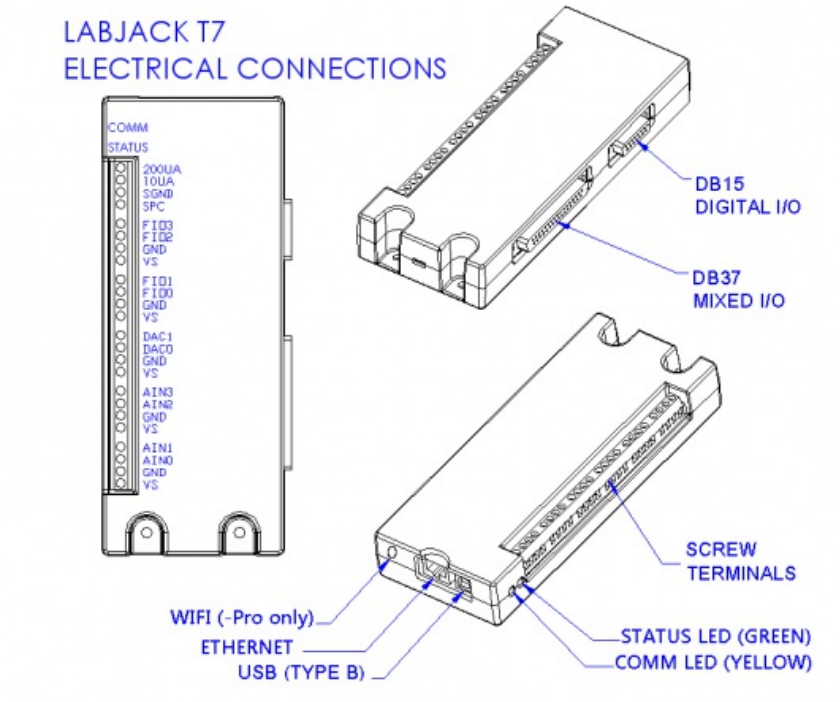
Appendix B - Enclosure and PCB Drawings

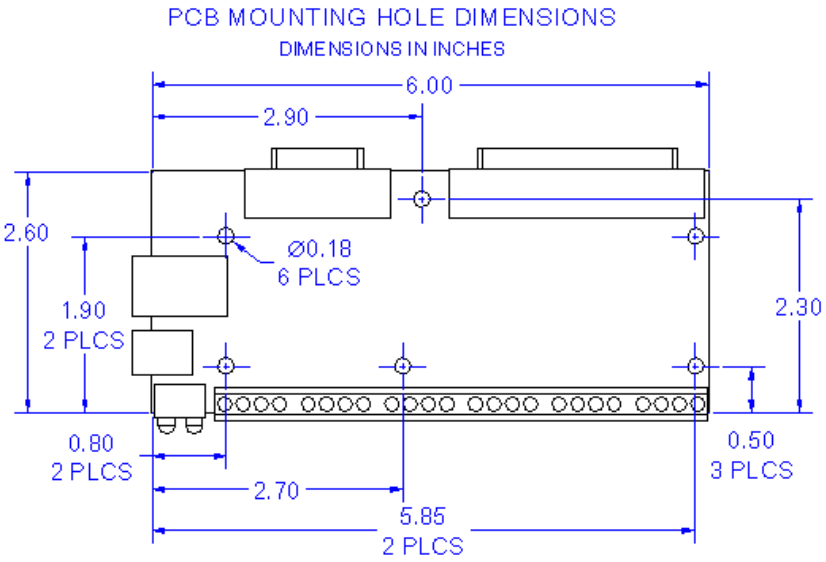
See below drawings of the T7.

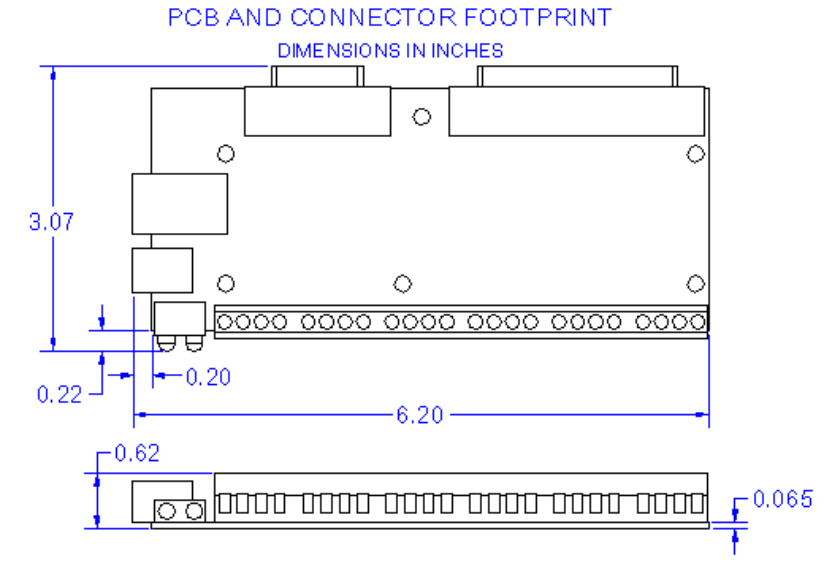
The square holes on the back of the enclosure are for DIN rail mounting adapters (TE Connectivity(formerly Tyco) part #TKAD), or Newark PN 50F2979.

CAD drawings of the T7 enclosure are attached to the bottom of this page. (DWG, DXF, IGES, STEP)




LABJACK T7
ELECTRICAL CONNECTIONS
COMM
STATUS
200UA
10UA
SGND
SPC
FIO3
FIO2
GND
VS
FIO1
FIO0
GND
VS
DAC1
DAC0
GND
VS
AIN3
AIN2
GND
VS
AIN1
AIN0
GND
VS
DB15
DIGITAL I/O
DB37
MIXED I/O
SCREW
TERMINALS
WIFI (-Pro only)
ETHERNET
USB (TYPE B)
STATUS LED (GREEN)
COMM LED (YELLOW)


PCB MOUNTING HOLE DIMENSIONS
DIMENSIONS IN INCHES
6.00
2.90
2.60
Ø0.18
6 PLCS
1.90
2 PLCS
2.30
0.80
2 PLCS
0.50
3 PLCS
2.70
5.85
2 PLCS

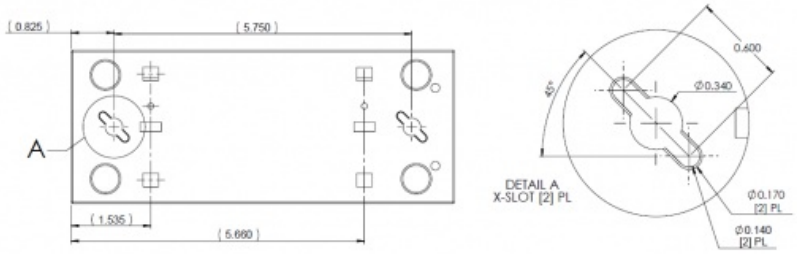

PCB AND CONNECTOR FOOTPRINT
DIMENSIONS IN INCHES
3.07
0.22
0.20
6.20
0.62
0.065

LabJack T7 Enclosure

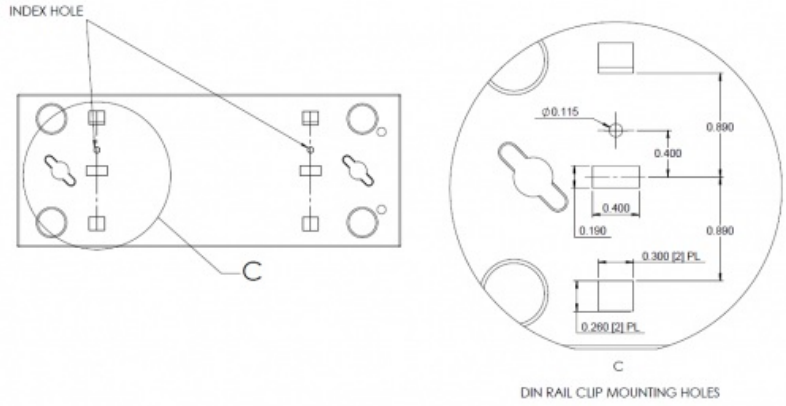
6-2-2012

Note: Dimensions in parentheses are reference only.

T7 ENCLOSURE BASE

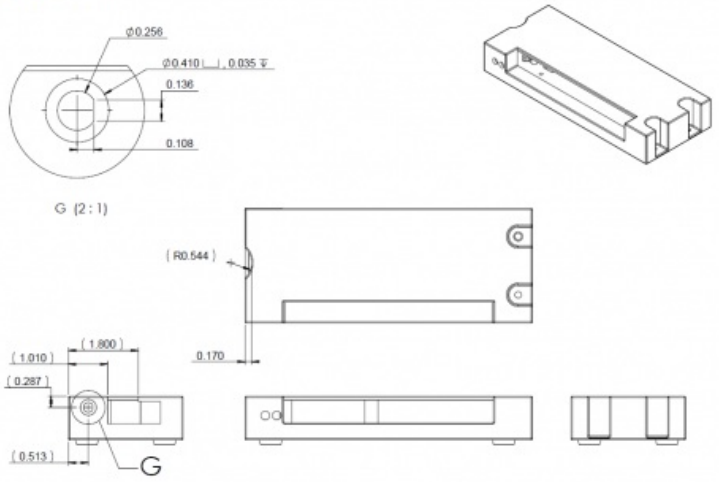


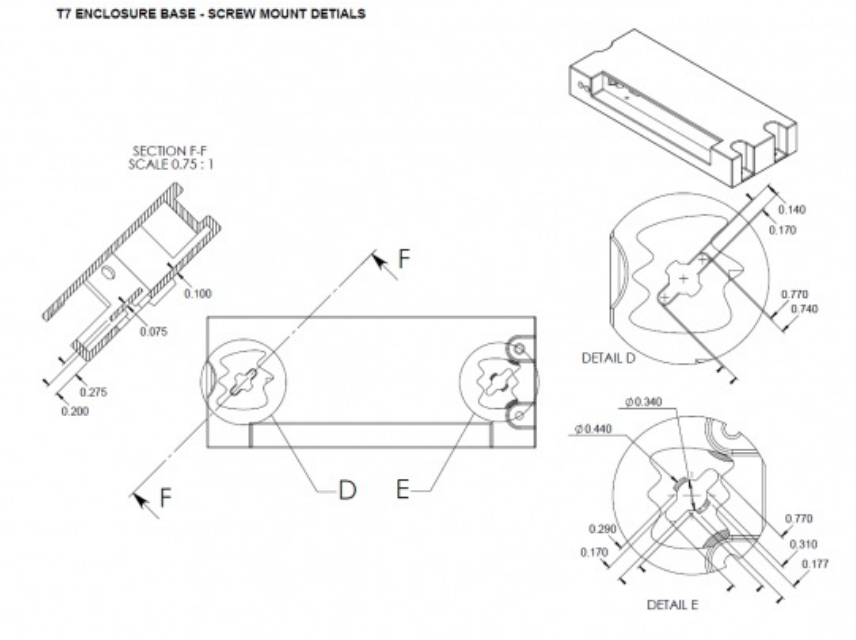
T7 ENCLOSURE BASE - DIN CLIP DETAILS



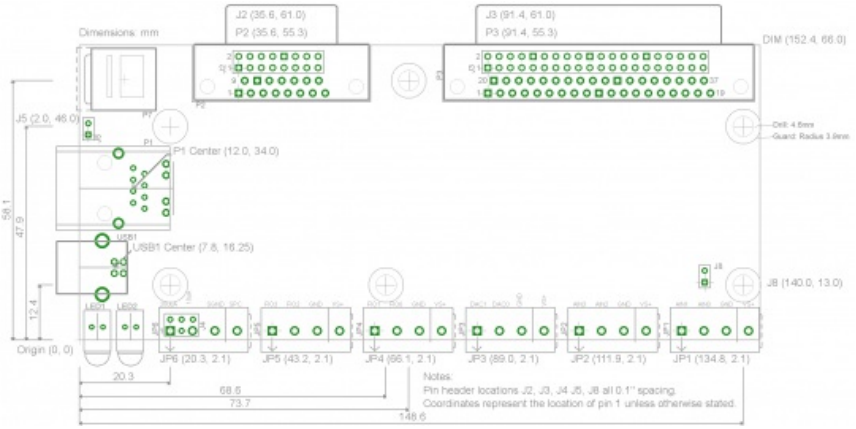
DIN RAIL CLIP MOUNTING HOLES

T7 ENCLOSURE TOP
CLEARANCE FOR SHROWDED ETHERNET CABLE
ANTENNA NUT BOSS DETIALS





T7 OEM PCB Dimensions



File attachment:

- T7 Enclosure DWG
- T7 Enclosure DXF
- T7 Enclosure IGS
- T7 PCB Hole Dimensions
- T7 PCB Connector Dimensions
- T7 Enclosure STEP
- T7 Pro Enclosure DWG
- T7 Pro Enclosure DXF
- T7 Pro Enclosure IGS
- T7 Pro Enclosure STEP
- T7 Mounting Hole Template

Appendix C - Firmware Revision History

The release T7 firmware is listed on the T7 firmware page. You will need the Kipling software program to load the firmware files onto a T7. Also use Kipling to identify the current WiFi and Firmware versions on your T7.

Change Log

1.0146: Fixed a bug that would prevent UART from being disabled once enabled. Also fixed a bug involving AIN_EF power-up default settings. On previous versions of firmware, configuring some AIN_EF channels as theromocouples, and then saving those settings as

power-up defaults would cause the device to enter a continuously-resetting state on the next power reset wherein the LEDs would blink rapidly. It is possible to recover from the continuously-resetting state by jumpering a wire between SPC and FIO3, or between SPC and FIO2.

Click To Expand Change Log

Appendix D - Packaging Information

Package Contents:

The normal retail packaged T7 or T7-Pro consists of:

- T7 (-Pro) unit itself in red enclosure
- USB cable (6ft / 1.8m)
- Ethernet Cable (6ft / 1.8m)
- USB 5V power supply
- Screwdriver
- Antenna (T7-Pro only)

Other package details:

There is no software CD included, so an internet connection is required to download software. Go to the T7 Support Homepage (labjack.com/support/t7) to get started.

Contact support@labjack.com for additional information on shipping.

Package size: 10" x 7" x 3"
Package wt: 1.2lb

Appendix E - Datasheet Revision History

Revision G (July 2014) Changed the modes for the Interrupt Counter with Debounce Extended Feature for consistency with firmware 1.0119 and greater. Small change to the text in the hardware overview block diagram. Added pictures to the SD card section. Added AIN_ALL_EF_INDEX to the list of registers in analog extended features. Updated a few other extended feature descriptions.

Revision F (April 2014) Added LED power registers to the LED section. Changed names of I2C and SPI registers from

READ/WRITE to RX/TX. Added WiFi network topology figures for connecting to an access point and controlling with a host. Added WiFi signal strength vs. distance from receiver. RTCC (real-time clock and calendar) section relabeled as RTC. Some updates to Appendix A.

Revision E (Feb 2014) Added sections for Flash Memory, SD Card, Stream-Out, and SPI. The internal temperature sensor section now has complete information. Also updated a few descriptions to registers. Moved SPI, I2C, and Digital I/O Extended Features into Digital I/O. Moved many sections into appendices - most notably the specifications page is now Appendix A.

Revision D (Jan 2014) Added Scripting, I2C sections, updated many sections to include register information directly from the constants file. Updated DIO_EF information.

Revision C (Oct 2013) Added calibration constants information. Modified URLs. Updated many links to related support material. Updated DIO information.

Revision B (April 2013) Added many descriptions of Digital I/O extended features. Modified a bunch of URLs.

Revision A (February 2013) Original data sheet for the T7 family of devices.